Developers Guide

Domail server

Nuncio j.s.a.

Contents

List of Figures

1 SMTP

What is SMTP? Simple Mail Transfer Protocol (SMTP) is a quick and easy way to send email from one server to another.

How is SMTP different from other email protocols? The main difference between these protocols is that SMTP is the only protocol for sending or pushing email from one unknown mail server to another.
POP and IMAP are protocols for receiving or pulling mail for the recipient from their own mail server.

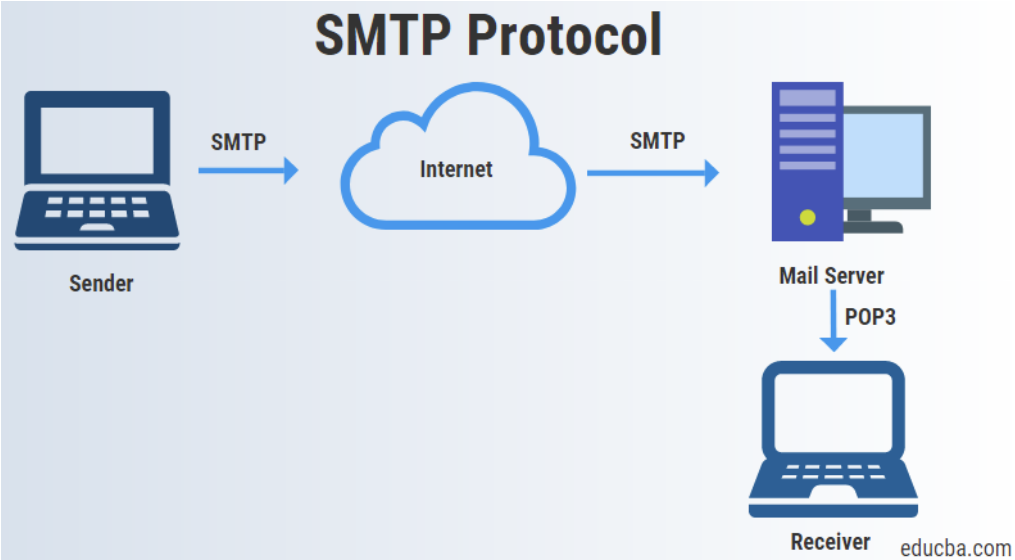

Figure 1: SMTP protocol

By default, SMTP to send email lacks encryption and can be used for sending without any protection in place, leaving emails with an SMTP setup susceptible to man-in-the-middle attacks and eavesdropping from bad actors while messages are in transit. SMTPS uses additional SSL or TLS cryptographic protocols for improved security, and the extra "S" stands for SECURE!
Secure SMTP can be achieved through the enablement of TLS on your mail server. By enabling TLS, you are encrypting the SMTP protocol on the transport layer by wrapping SMTP inside of a TLS connection. This effectively secures SMTP and transforms it into SMTPS.

Port 587 and 465 are both frequently used for SMTPS traffic. Port 587 is often used to encrypt SMTP messages using STARTTLS, which allows the email client to establish secure connections by requesting that the mail server upgrade the connection through TLS.
Port 465 is used for implicit TLS and can be used to facilitate secure communications for mail services. According to the Internet Engineering Task Force, or IETF, this is preferred over using STARTTLS on port 587.
Lastly, port 2525 is sometimes also used. Some residential ISPs will block port 25 to stop users from running their own mail servers. To combat this, enthusiasts and

small home businesses use port 2525.
SMTPS plays a key role in email security, but it can't protect against all email-based threats.

1.1 Example of sendig email through SMTP

This example shows how to send an email via SMTP without using any encryption.

1.1.1 C#

First language is C#, in which we used SmtpClient for sending email. Complete source file is on the next listening 1 or the whole solution with examples can be downloaded from: TODO: WWW

```
class EmailSmtp
{
    private MailMessage GetMessage(string bodyMessage = "") {

        MailAddress from = new MailAddress(Configuration.MailFrom);
        MailAddress to = new MailAddress(Configuration.MailTo);

        MailMessage message = new MailMessage(from, to);
        message.Subject = Configuration.MailSubject;
        message.Body = string.IsNullOrEmpty(bodyMessage) ? $"Test SMTP body from C#" : bodyMessage;
        message.IsBodyHtml = true;
        message.Priority = MailPriority.Normal;

        return message;
    }

    /// <summary>
    /// Send email without encryption
    /// </summary>
    public void ScriptExampleSmtp_EmailSend1() {
        try {
            SmtpClient client = new SmtpClient();
            client.Host = Configuration.URL;
            client.Port = 25;
            client.EnableSsl = false;

            var message = GetMessage();
            client.Send(message);
        }
        catch (SmtpException ex) {
            Console.WriteLine($"SMTPEXCEPTION: {ex}");
        }
        catch (Exception ex) {
            Console.WriteLine($"Exception: {ex}");
        }
    }
}
```

Listing 1: Example for sending email through SMTP in C#

1.1.2 Java

TODO

1.1.3 Python

TODO

1.2 Example of sendig email through SMTP with TLS

This example shows how to send an email via SMTP with TLS.

1.2.1 C#

First language is C#, complete source file is on the next listening 2 or the whole solution with examples can be downloaded from: TODO: WWW

```
class EmailSmtp
{
    private MailMessage GetMessage(string bodyMessage = "") {

        MailAddress from = new MailAddress(Configuration.MailFrom);
        MailAddress to = new MailAddress(Configuration.MailTo);

        MailMessage message = new MailMessage(from, to);
        message.Subject = Configuration.MailSubject;
        message.Body = string.IsNullOrEmpty(bodyMessage) ? $"Test SMTP body from C#" :
     bodyMessage;
        message.IsBodyHtml = true;
        message.Priority = MailPriority.Normal;

        return message;
    }





}
```

Listing 2: Example for sending email through SMTP with TLS in C#

1.2.2 Java

TODO

1.2.3 Python

TODO

1.3 Example of sendig email through SMTP with SSL

This example shows how to send an email via SMTP with SSL.

1.3.1 C#

First language is C#, complete source file is on the next listening 3 or the whole solution with examples can be downloaded from: TODO: WWW

```
class EmailSmtp
{
    private MailMessage GetMessage(string bodyMessage = "") {

        MailAddress from = new MailAddress(Configuration.MailFrom);
        MailAddress to = new MailAddress(Configuration.MailTo);

        MailMessage message = new MailMessage(from, to);
        message.Subject = Configuration.MailSubject;
        message.Body = string.IsNullOrEmpty(bodyMessage) ? $"Test SMTP body from C#" : bodyMessage;
        message.IsBodyHtml = true;
        message.Priority = MailPriority.Normal;

        return message;
    }





}
```

Listing 3: Example for sending email through SMTP with SSL in C#

1.3.2 Java

TODO

1.3.3 Python

TODO

- Popis protokolu - Podporovane protokoly: âŮŃ SSL/TLS âĂę- Popisat spracovanie emailu - ked sa odosiela email - Script vyberie plain text,

2 SOAP

WSDL can be downloaded from the Domail website:
http://{domail_address}/domail/domail.wsdl

domail_Communication		
_create		
input	partCreateCommunicationRequest	createCommunicationRequest
output	partCreateCommunicationResponse	communicationResponse
_createFromTemplate		
input	partCreateCommunicationFromTemplateRequest	createCommunicationFromTemplate
output	response	EmailSendWithTemplateResponse
emailSend		
input	emailSendRequest	emailSendRequestElement
output	response	communicationResponse
emailSendAdv		
input	emailSendAdvRequest	emailSendAdvRequest
output	response	communicationResponse
communicationGet		
input	communicationGetRequest	communicationGetRequest
output	response	communicationGetResponse
communicationGetState		
input	communicationGetStateRequest	communicationGetRequest
output	response	communicationGetStateResponse
emailSendWithTemplateAdv		
input	emailSendWithTemplateAdvRequest	emailSendWithTemplateAdvRequest
output	response	emailSendWithTemplateResponse
emailSendWithTemplate		
input	emailSendWithTemplateRequest	emailSendWithTemplateRequest
output	response	emailSendWithTemplateResponse

Figure 2: SOAP Documentation for DoMail server

2.1 Definition calls

The following endpoints are available in communication:

- emailSend
- emailSendAdv
- communicationGet
- communicationGeteState
- emailSendWithTemplateAdv
- emailSendWithTemplate

2.1.1 emailSend

- Endpoint is used to simply send an email via SOAP channel
- Body of POST request is XML object EmailSendRequestType

In the following figure 3 is the definition of the request XML object 'EmailSendRequestType':

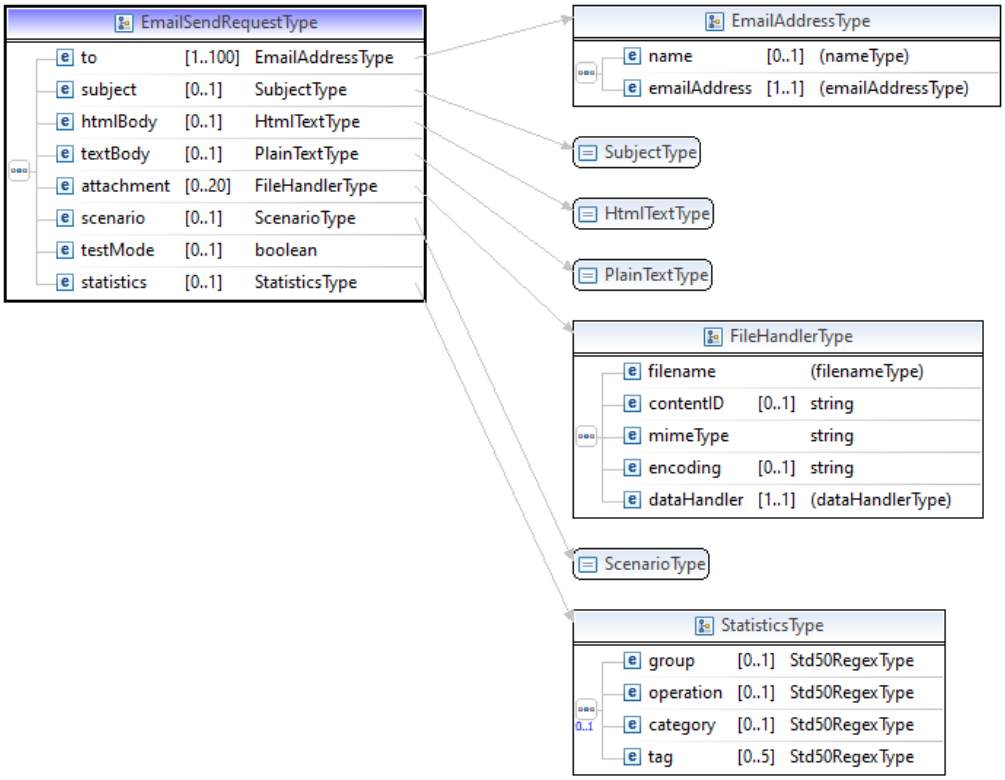

Figure 3: SOAP definition for type 'EmailSendRequestType'

- **to** (required field)
  - list of names and email address of recipients
  - minimum count of recipients is 1 and maximum 100 recipients
- **subject**
  - subject of email
  - maximum length is 255 characters
- **htmlBody**
  - html text body of email
  - string
- **textBody**

  - plain text body of email
  - string
- **attachments**
  - array of attachments
  - maximum count of items is 20
  - One item of type **fileHandlerType**:
    - **filename** (required field)
      - file name
      - maximum length is 255 characters
    - **contentId**
      - Content ID
      - string
    - **mimeType**
      - MimeType (Example: application/text)
      - string
    - **encoding**
      - Encoding (Example: utf-8)
      - string
    - **dataHandler** (required field)
      - Base64 string
- **scenario** (required field)
  - the name of the script to be used for this communication
  - pattern: [-0-9a-zA-Z_@#/́]+
  - maximum length is 255 characters
- **testmode**
  - if the field is set to TRUE, the communication is in test mode
  - default value is FALSE
- **statistics**
  - statistical data that help to classify a given communication
  - **group**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **operation**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **category**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters

- **tags**
  - maximum count of items is 5

In the following figure 12 is the definition of the response XML object 'CommunicationResponseType':

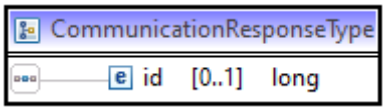

Figure 4: SOAP definition for type 'CommunicationResponseType'

Response XML object 'CommunicationResponseType':

- **id**
  - Id of send communication in the system Domail,
  - It has to be bigger than 0

2.1.2 emailSendAdv

In the following figure 5 is the definition of the request XML object 'EmailSendAdvRequestType':

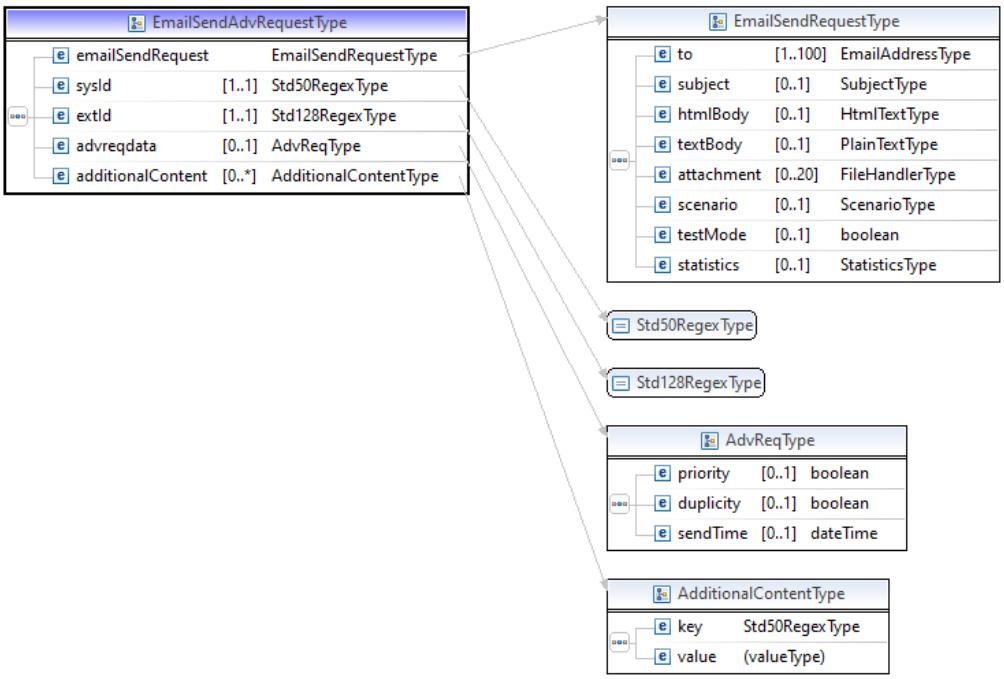

Figure 5: SOAP definition for type 'EmailSendAdvRequestType'

XML type 'EmailSendAdvRequestType' contains also property emailSendRequest - type 'EmailSendRequestType' and continues with new fields:

- **sysId**

  - the unique id/name of the external system wich use this the system Domail
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 50 characters

- **extId**
  - the unique id in external system wich use this the system Domail.
  - the Domail checks if duplicity is enabled/disabled. If 'duplicity' is disabled so it will not send more emails with the same 'extId'
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 128 characters

- **advreqdata** - object type is 'AdvdReqType', subfields are:
  - **priority**
    * if we want to send a high priority email, we set the value to TRUE
    * boolean
    * value is true/false
  - **duplicity**
    * if we want to send more emails with the same extId, we have to set the value to TRUE
    * boolean
    * value is true/false
  - **sendTime**
    * If it is set, the email will be sent at that time
    * string (date-time)
    * value have to be time greater than now

- **additionalContent** - array of objects type 'AdditionalContentType'. Subfields of object type 'additionalContentType' are:
  - **key**
    * Key is unique name of item
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **value**
    * Value of item
    * object

Response is the same as in the case of the EmailSendSimple call (figure 6)

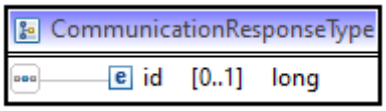

Figure 6: SOAP definition for type 'CommunicationResponseType'

Response XML object 'CommunicationResponseType':

- **id**
  - Id of send communication in the system Domail,
  - It has to be bigger than 0

2.1.3 communicationGet

Get details of given communication request by Id or extId. In the following figure 7 is the definition of the request XML object 'CommunicationGetRequestType':

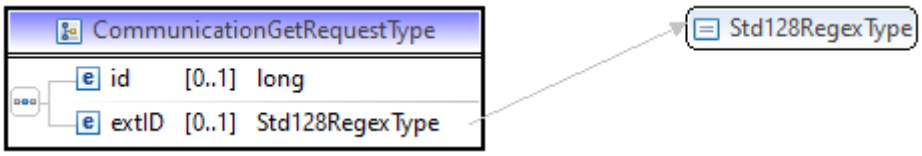

Figure 7: SOAP definition for type 'CommunicationGetRequestType'

The input parameter is the id or extId. In the following figure 8 is the return value when a request is successfully sent to the Domail system. It is the definition of the response XML object 'CommunicationDetailType':

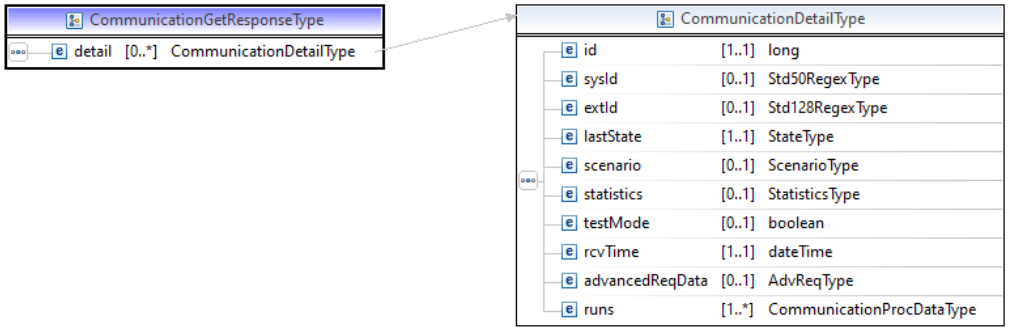

Figure 8: SOAP definition for type 'CommunicationDetailType'

Object of type 'CommunicationDetailType' has fields:

- **id** (required field)
  - main id of communication
- **sysId**
  - the unique id/name of the external system wich use this the system Domail
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 50 characters
- **extId**
  - the unique id in external system wich use this the system Domail.
  - the Domail checks if duplicity is enabled/disabled. If 'duplicity' is disabled so it will not send more emails with the same 'extId'

  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 128 characters

- **lastState**
  - **processingState**
    * last processing status
    * string
    * one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE,
      PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
  - **deliveryState**
    * last delivery status
    * string
    * one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}

- **scenario**
  - the name of the script to be used for this communication
  - pattern: [-0-9a-zA-Z_@#/́]+
  - maximum length is 255 characters

- **statistics**
  - statistical data that help to classify a given communication
  - **group**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **operation**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **category**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **tags**
    * maximum count of items is 5

- **testmode**
  - if the field is set to TRUE, the communication is in test mode
  - default value is FALSE

- **rcvTime**
  - time of receipt of the communication

- **advancedReqData** - object type is 'AdvReqType', subfields are:
  - **priority**
    - if we want to send a high priority email, we set the value to TRUE
    - boolean
    - value is true/false
  - **duplicity**
    - if we want to send more emails with the same extId, we have to set the value to TRUE
    - boolean
    - value is true/false
  - **sendTime**
    - If it is set, the email will be sent at that time
    - string (date-time)
- **runs** - object type is 'CommunicationProcDataType', subfields are:
  - **runId**
    - running id of communication
    - integer
    - value have to be greater as 0
  - **procTime**
    - date and time of processing
    - string (date-time)
  - **scenarioName**
    - the name of the script was used for this communication
    - maximum length is 255 characters
  - **processingState**
    - processing status
    - string
    - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE,
      PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
  - **deliveryState**
    - last delivery status
    - string
    - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
  - **sentEmails**
    - more emails can be generated during the processing of the communication, here are the details of each
    - object type is 'SendCommunicationType'

- subfields are:
- **addressNumber** (required field)
  - email number in communication processing
  - integer
- **emailAddress**
  - email address
  - pattern: `[^@]+@[^.]+..+`
  - maximum length is 512 characters
- **processingState**
  - processing status
  - string
  - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED,
    DISPATCH_ERROR, CANCELLED}
- **deliveryState**
  - last delivery status
  - string
  - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
- **messageId**
  - message ID
  - maximum length is 80 characters
- **sendTime**
  - If it is set, the email will be sent at that time
  - string (date-time)

2.1.4 communicationGetState

In the following figure 9 is the definition of the request XML object 'CommunicationGetStateRequestType':

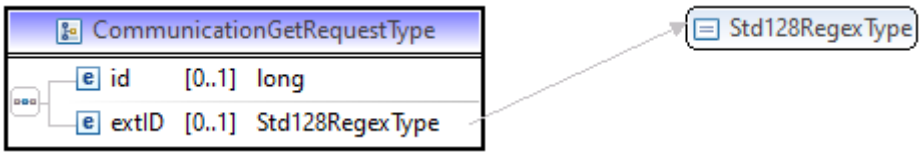

Figure 9: SOAP definition for type 'CommunicationGetRequestType'

The input parameter is the id or extId. In the following figure 10 is the definition of the response XML object 'CommunicationStateType':

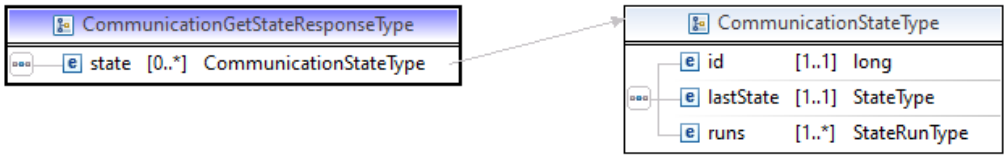

Figure 10: SOAP definition for type 'CommunicationStateType'

Object of type 'CommunicationStateType' has fields:

- **id** (required field)
  - main id of communication
- **lastState**
  - **processingState**
    - last processing status
    - string
    - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
  - **deliveryState**
    - last delivery status
    - string
    - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
- **runs** - object type is 'StateRunType', subfields are:
  - **runId**
    - running id of communication
    - integer
    - value have to be greater as 0
  - **procTime**
    - date and time of processing
    - string (date-time)
  - **scenarioName**
    - the name of the script was used for this communication
    - maximum length is 255 characters
  - **processingState**
    - processing status
    - string
    - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}

- **deliveryState**
  - last delivery status
  - string
  - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
- **sentEmails**
  - more emails can be generated during the processing of the communication, here are the details of each
  - object type is 'SendCommunicationType'
  - subfields are:
  - **addressNumber** (required field)
    - email number in communication processing
    - integer
  - **emailAddress**
    - email address
    - pattern: `[^@]+@[^.]+..+`
    - maximum length is 512 characters
  - **processingState**
    - processing status
    - string
    - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
  - **deliveryState**
    - last delivery status
    - string
    - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
  - **messageId**
    - message ID
    - maximum length is 80 characters
  - **sendTime**
    - If it is set, the email will be sent at that time
    - string (date-time)

2.1.5 emailSendWithTemplate

In the following figure 11 is the definition of the request XML object 'EmailSendWithTemplateRequestType':

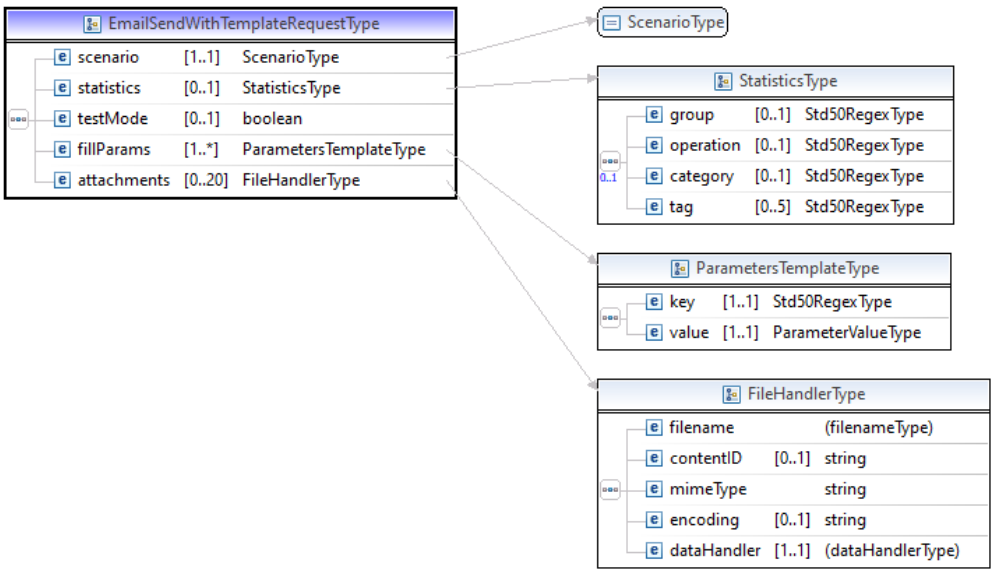

Figure 11: SOAP definition for type 'EmailSendWithTemplateRequestType'

Properties of object type 'EmailSendWithTemplateRequestType':

- **scenario** (required field)
  - the name of the script to be used for this communication
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 255 characters
- **statistics** - JSON object type of 'StatisticsType'
  - statistical data that help to classify a given communication
  - **group**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **operation**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **category**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **tags**
    - maximum count of items is 5
    - pattern: [-0-9a-zA-Z_@#/]+
- **testmode**
  - if the field is set to TRUE, the communication will be to send in test mode
  - default value is FALSE

- **params**
  - list of parameters to be used in the template
  - array of objects type of 'ParametersTemplateType'
  - **key**
    - string
    - pattern: [-0-9a-zA-Z_@#/]+
  - **value**
    - string
    - maximum length is 1024 characters
- **attachments**
  - array of attachments
  - maximum count of items is 20
  - One item of type **fileHandlerType**:
    - **filename** (required field)
      - file name
      - maximum length is 255 characters
    - **contentId**
      - Content ID
      - string
    - **mimeType**
      - MimeType (Example: application/text)
      - string
    - **encoding**
      - Encoding (Example: utf-8)
      - string
    - **dataHandler** (required field)
      - Base64 string

In the following figure 12 is the definition of the response XML object 'EmailSendWithTemplateResponseType':

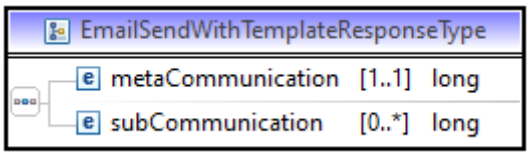

Figure 12: SOAP definition for type 'EmailSendWithTemplateResponseType'

- **metaCommunication** (required field)
  - main id of communication
- **subCommunication**
  - list of id of communications
  - minimum count is 0

2.1.6 emailSendWithTemplateAdv

In the following figure 13 is the definition of the request XML object 'EmailSendWithTemplateAdvRequestType':

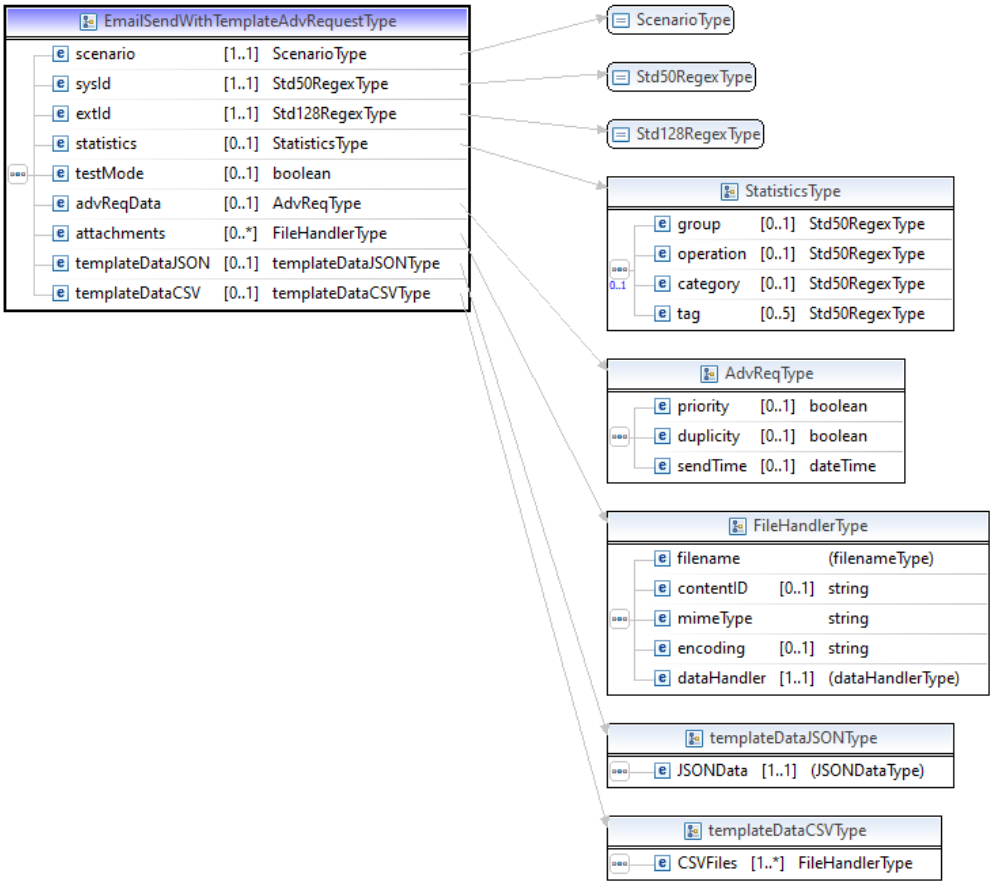

Figure 13: SOAP definition for type 'EmailSendWithTemplateAdvRequestType'

Properties of object type 'EmailSendWithTemplateAdvRequestType':

- **scenario** (required field)
  - the name of the script to be used for this communication
  - pattern: [-0-9a-zA-Z_@#/́]+
  - maximum length is 255 characters
- **sysId** (required field)
  - the unique id/name of the external system wich use this the system Domail
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 50 characters
- **extId** (required field)

  - the unique id in external system wich use this the system Domail.
  - the Domail checks if duplicity is enabled/disabled. If 'duplicity' is disabled so it will not send more emails with the same 'extId'
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 128 characters

- **statistics** - JSON object type of 'StatisticsType'
  - statistical data that help to classify a given communication
  - **group**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **operation**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **category**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **tag**
    * maximum count of items is 5
    * pattern: [-0-9a-zA-Z_@#/]+
- **testMode**
  - if the field is set to TRUE, the communication will be to send in test mode
  - default value is FALSE
- **advRequData** - object type is 'AdvReqType', subfields are:
  - **priority**
    * if we want to send a high priority email, we set the value to TRUE
    * boolean
    * value is true/false
  - **duplicity**
    * if we want to send more emails with the same extId, we have to set the value to TRUE
    * boolean
    * value is true/false
  - **sendTime**
    * If it is set, the email will be sent at that time
    * string (date-time)
    * value have to be time greater than now
- **attachments**
  - array of attachments

  - maximum count of items is 20
  - One item of type **FileHandlerType**:
    * **filename** (required field)
      · file name
      · maximum length is 255 characters
    * **contentId**
      · Content ID
      · string
    * **mimeType**
      · MimeType (Example: application/text)
      · string
    * **encoding**
      · Encoding (Example: utf-8)
      · string
    * **dataHandler** (required field)
      · Base64 string

- **templateDataJSON**
  - array of any objects
- **templateDataCSV**
  - object of type **fileHandlerType** - same as was in attachments:
    * **filename** (required field)
      · file name
      · maximum length is 255 characters
    * **contentId**
      · Content ID
      · string
    * **mimeType**
      · MimeType (Example: application/text)
      · string
    * **encoding**
      · Encoding (Example: utf-8)
      · string
    * **dataHandler** (required field)
      · Base64 string

In the following figure 14 is the definition of the response XML object 'EmailSendWithTemplateResponseType':

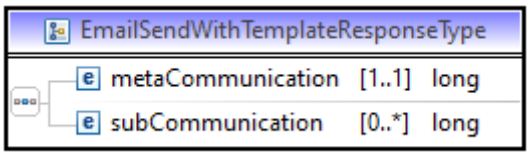

Figure 14: SOAP definition for type 'EmailSendWithTemplateResponseType'

- **metaCommunication** (required field)
  - main id of communication
- **subCommunication**
  - list of id of communications
  - minimum count is 0

2.2 Examples

This chapter contains examples for individual SOAP calls from the previous chapter.

2.2.1 Example for emailSend

This is followed by a SOAP request in XML (4), which secures the sending of the email via the EmailSend call.

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:p1="urn:dominanz.sk/domail">
    <soap:Header></soap:Header>
    <soap:Body>
        <p1:emailSendRequestElement>
        <to>
            <name>lukas</name>
            <emailAddress>lukas.vanek+soap@dominanz.sk</emailAddress>
        </to>
        <subject>Test SOAP email</subject>
        <scenario>Lukas-SOAP/REST</scenario>
    </p1:emailSendRequestElement>
    </soap:Body>
</soap:Envelope>
```

Listing 4: SOAP request for EmailSend simple

After sending the request to the server, you can see mode details from tab General 15 and EML source in 16 in the communication detail

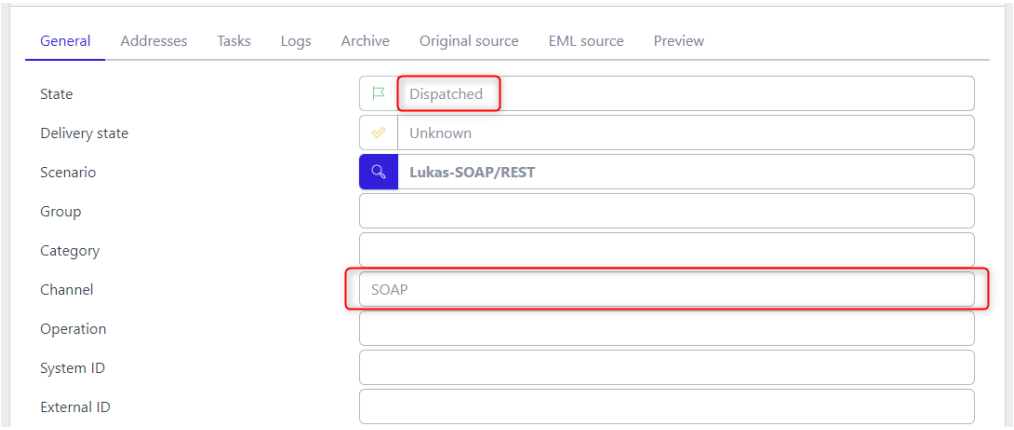

Figure 15: CommunicationDetail for SOAP Request for EmailSend simple

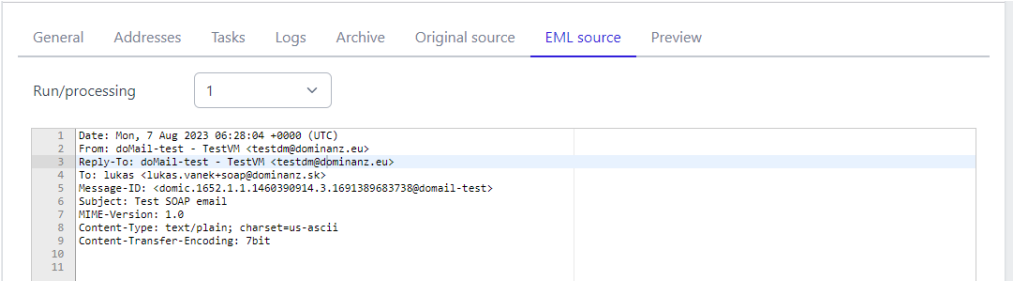

Figure 16: CommunicationDetail for SOAP Request for EmailSend simple - EML

2.2.2 Example for emailSendAdv

This is followed by a SOAP request in XML (5), which secures the sending of the email via the EmailSendAdv call. The difference from simple is e.g. SysId and ExtId.

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:p1="urn:dominanz.sk/domail">
    <soap:Header></soap:Header>
    <soap:Body>
    <p1:emailSendAdvRequest>
        <emailSendRequest>
            <to>
                <name>lukas</name>
                <emailAddress>lukas.vanek+soap@dominanz.sk</emailAddress>
            </to>
            <scenario>Lukas-SOAP/REST</scenario>
        </emailSendRequest>
        <sysId>34242</sysId>
        <extId>0001</extId>
    </p1:emailSendAdvRequest>
    </soap:Body>
</soap:Envelope>
```

Listing 5: SOAP request for EmailSendAdv

After sending the request to the server, you can see mode details from tab General 17 and EML source in 18 in the communication detail

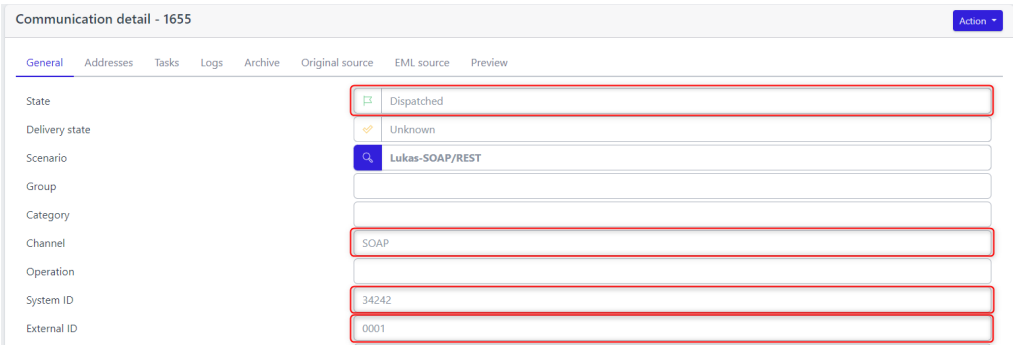

Figure 17: CommunicationDetail for SOAP Request for EmailSendAdv

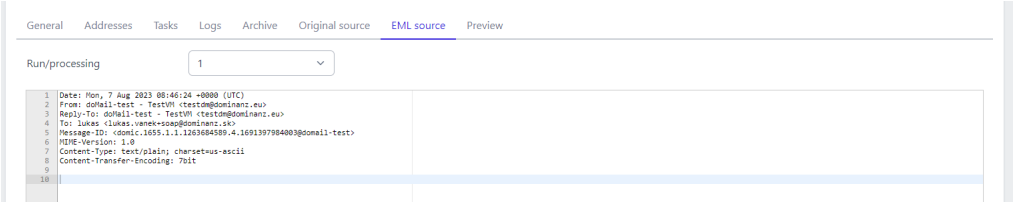

Figure 18: CommunicationDetail for SOAP Request for EmailSendAdv - EML

If we send the same SOAP request 5 to the server again, we would get the following response 6. If we do not change the ExtId and the duplicity check is enabled, an error occurs during processing and the email communication is not sent (19 and 20).

```
    <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns3:communicationResponse xmlns:ns3="urn:dominanz.sk/domail">
            <id>1656</id>
        </ns3:communicationResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 6: SOAP response for EmailSendAdv

Id	Address	Processing	Delivery	Scenario	Group	Category	Send time
1656	lukas.vanek+soap@dominanz.sk			Lukas-SOAP/REST			07.08.2023 08:49:20
1655	lukas.vanek+soap@dominanz.sk			Lukas-SOAP/REST			07.08.2023 08:46:23

Figure 19: List of communications for the same SOAP Request for EmailSendAdv

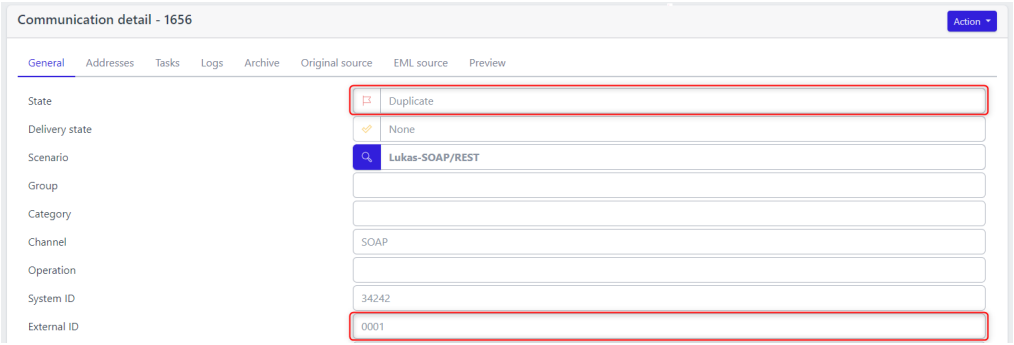

Figure 20: Second communicationDetail for SOAP Request for EmailSendAdv

2.2.3 Example for communicationGet

The following code 7 contains an example of calling CommunicationDetail for a non-existent id

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:p1="urn:dominanz.sk/domail">
  <soap:Header></soap:Header>
```

```
  <soap:Body>
    <p1:communicationGetRequest>
        <id>1660</id>
    </p1:communicationGetRequest>
  </soap:Body>
</soap:Envelope>
```

Listing 7: SOAP request for CommunicationDetail - not found for id

If the communication with the given ID is not found, the following response 8 is received:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
    <SOAP-ENV:Fault>
        <faultcode>SOAP-ENV:Server</faultcode>
        <faultstring xml:lang="en">No request found for id = 1660</faultstring>
    </SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 8: SOAP response for CommunicationDetail - not found for id

The following code 9 contains an example of calling CommunicationDetail for a existent id:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:p1="urn:dominanz.sk/domail">
    <soap:Header></soap:Header>
    <soap:Body>
    <p1:communicationGetRequest>
        <id>1655</id>
    </p1:communicationGetRequest>
    </soap:Body>
</soap:Envelope>
```

Listing 9: SOAP request for CommunicationDetail

If the communication with the given ID is found, the following response 10 is received:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns3:communicationGetResponse xmlns:ns3="urn:dominanz.sk/domail">
            <detail>
                <id>1655</id>
                <sysId>34242</sysId>
                <extId>0001</extId>
                <lastState>
                    <processingState>DISPATCHED</processingState>
                    <deliveryState>DELIVERY_UNKNOWN</deliveryState>
                </lastState>
                <scenario>Lukas-SOAP/REST</scenario>
                <statistics/>
                <testMode>false</testMode>
                <rcvTime>2023-08-07T08:46:23.725Z</rcvTime>
                <advancedReqData>
                    <priority>true</priority>
                    <duplicity>false</duplicity>
                    <sendTime>2023-08-07T08:46:23.725Z</sendTime>
```

```
                </advancedReqData>
                <runs>
                    <runId>1</runId>
                    <procTime>2023-08-07T08:46:23.733Z</procTime>
                    <state>
                        <processingState>DISPATCHED</processingState>
                        <deliveryState>DELIVERY_UNKNOWN</deliveryState>
                    </state>
                    <sentEmails>
                        <addressNumber>1</addressNumber>
                        <address>
                            <name>lukas</name>
                            <emailAddress>lukas.vanek+soap@dominanz.sk</emailAddress>
                        </address>
                        <state>
                            <processingState>DISPATCHED</processingState>
                            <deliveryState>DELIVERY_UNKNOWN</deliveryState>
                        </state>
                        <messageId>domic.1655.1.1.257431620.3.1691397984871@domail-test
                                </messageId>
                        <sendTime>2023-08-07T08:46:25.699Z</sendTime>
                    </sentEmails>
                </runs>
            </detail>
        </ns3:communicationGetResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 10: SOAP response for CommunicationDetail - all returned fields

2.2.4 Example of communicationGetState

The following code 11 contains an example of calling CommunicationGetState:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:p1="urn:dominanz.sk/domail">
    <soap:Header></soap:Header>
    <soap:Body>
    <p1:communicationGetStateRequest>
        <id>1655</id>
    </p1:communicationGetStateRequest>
    </soap:Body>
</soap:Envelope>
```

Listing 11: SOAP request for CommunicationGetState

If the communication with the given ID is found, the following response 12 is received:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns3:communicationGetStateResponse xmlns:ns3="urn:dominanz.sk/domail">
            <state>
                <id>1655</id>
                <lastState>
                    <processingState>DISPATCHED</processingState>
                    <deliveryState>DELIVERY_UNKNOWN</deliveryState>
                </lastState>
                <runs>
                    <runId>1</runId>
```

```
                    <state>
                        <processingState>DISPATCHED</processingState>
                        <deliveryState>DELIVERY_UNKNOWN</deliveryState>
                    </state>
                    <sentEmails>
                        <addressNumber>1</addressNumber>
                        <address>
                            <name>lukas</name>
                            <emailAddress>lukas.vanek+soap@dominanz.sk</emailAddress>
                        </address>
                        <state>
                            <processingState>DISPATCHED</processingState>
                            <deliveryState>DELIVERY_UNKNOWN</deliveryState>
                        </state>
                    </sentEmails>
                </runs>
            </state>
        </ns3:communicationGetStateResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 12: SOAP response for CommunicationGetState - all returned fields

2.2.5 Example of emailSendWithTemplate

To test further functionality we have prepared a template, which is called from the following scenario, which is then selected in the SOAP Request:

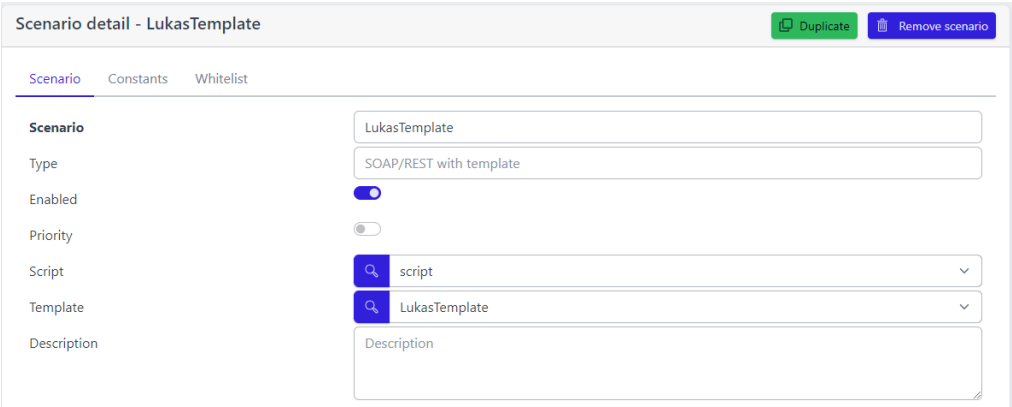

Figure 21: EmailSendWithTemplate - Scenario

In the following picture 22, we will define what variables will be used in the template.

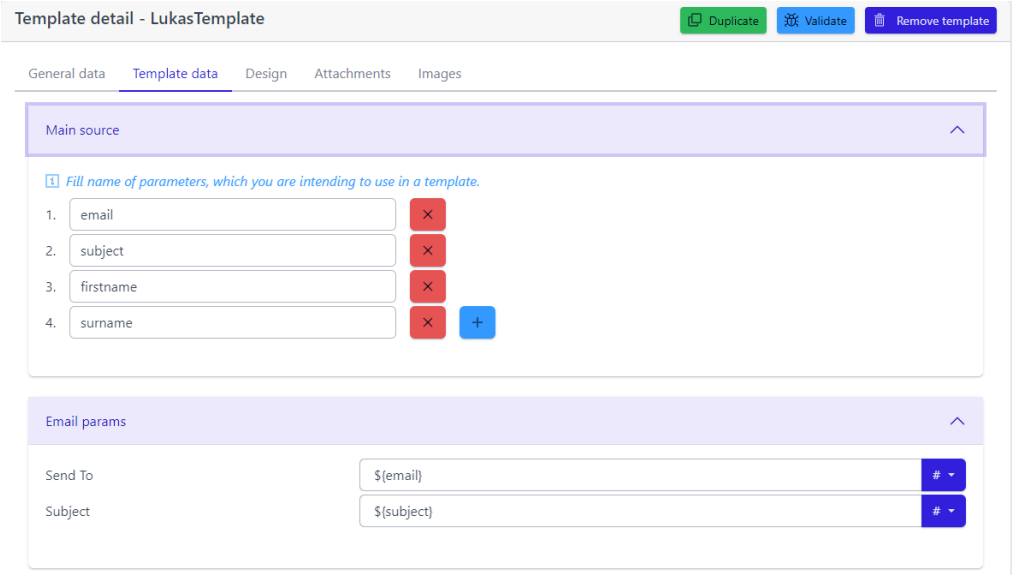

Figure 22: EmailSendWithTemplate - Template data

In the following picture 23, we will define the HTML points, where we will also use the prepared variables.

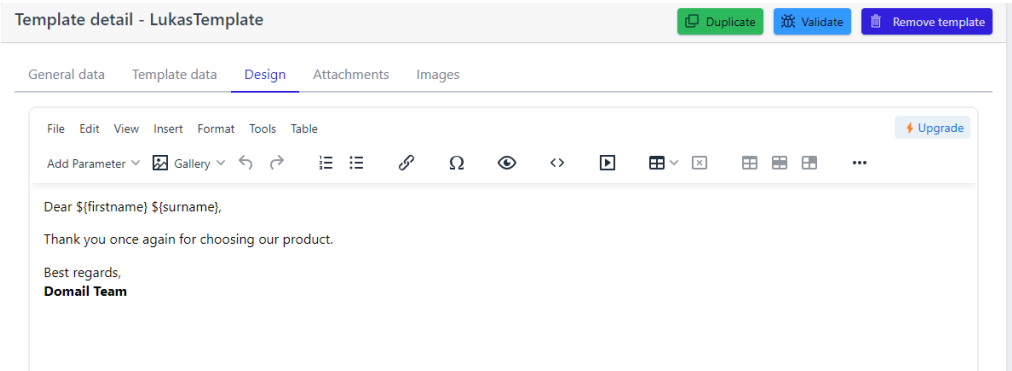

Figure 23: EmailSendWithTemplate - Design

The following code 13 contains an example of calling emailSendWithTemplate. In the example, variables that are defined in the template are also sent.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:p1="
    urn:dominanz.sk/domail">
    <soap:Header></soap:Header>
    <soap:Body>
    <p1:emailSendWithTemplateRequest>
        <scenario>LukasTemplate</scenario>
        <fillParams>
            <key>email</key>
            <value>lukas.vanek+soap1@dominanz.sk</value>
        </fillParams>
        <fillParams>
            <key>subject</key>
```

```
            <value>Subject of email</value>
        </fillParams>
        <fillParams>
            <key>firstname</key>
            <value>Lukas</value>
        </fillParams>
        <fillParams>
            <key>surname</key
            ><value>V.</value>
        </fillParams>
    </p1:emailSendWithTemplateRequest>
    </soap:Body>
</soap:Envelope>
```

Listing 13: SOAP request for emailSendWithTemplate

After a successful SOAP request is sent, a SOAP response 14 is returned with information about the metacommunication and subcommunication.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns3:emailSendWithTemplateResponse xmlns:ns3="urn:dominanz.sk/domail">
            <metaCommunication>1684</metaCommunication>
            <subCommunication>1685</subCommunication>
        </ns3:emailSendWithTemplateResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 14: SOAP response for emailSendWithTemplate

In the following picture 24, you can see 2 communications from the communication list.

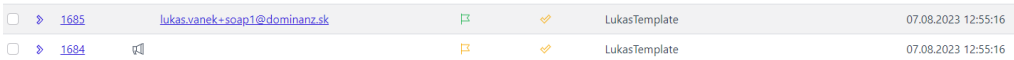

Figure 24: List of communications for SOAP Request for EmailSendWithTemplate

In the following picture 25, you can see a sub-communication in which you can see information about the delivery status and the selected template.

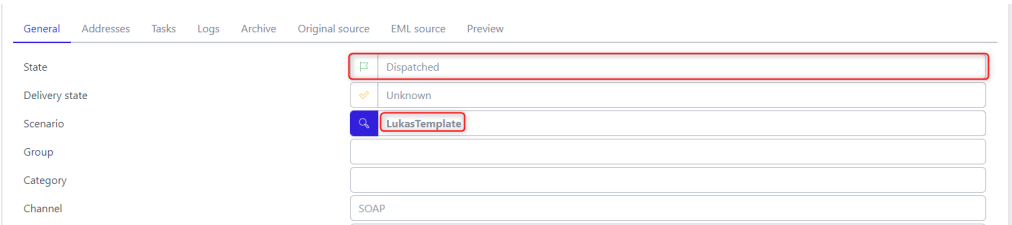

Figure 25: CommunicationDetail for SOAP Request for EmailSendWithTemplate - General tab

2.2.6 Example of emailSendWithTemplateAdv

In the following figure 26, a new scenario is prepared, where we can see the call 'EmailSendWithTemplateAdv'.

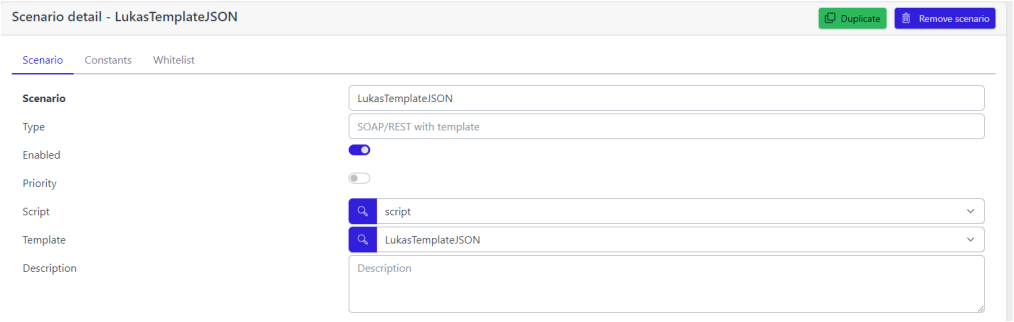

Figure 26: List of communications for SOAP Request for EmailSendWithTemplateAdv

After creating the template, select JSON as the Main source and import the following JSON:

```
[{ "email": "",
  "subject": "",
  "firstname" : "",
  "surname" : ""}
]
```

Listing 15: JSON file example for emailSendWithTemplateAdv

In the following figure 27, we can see parsed data in Main source and used first variables in Email params:

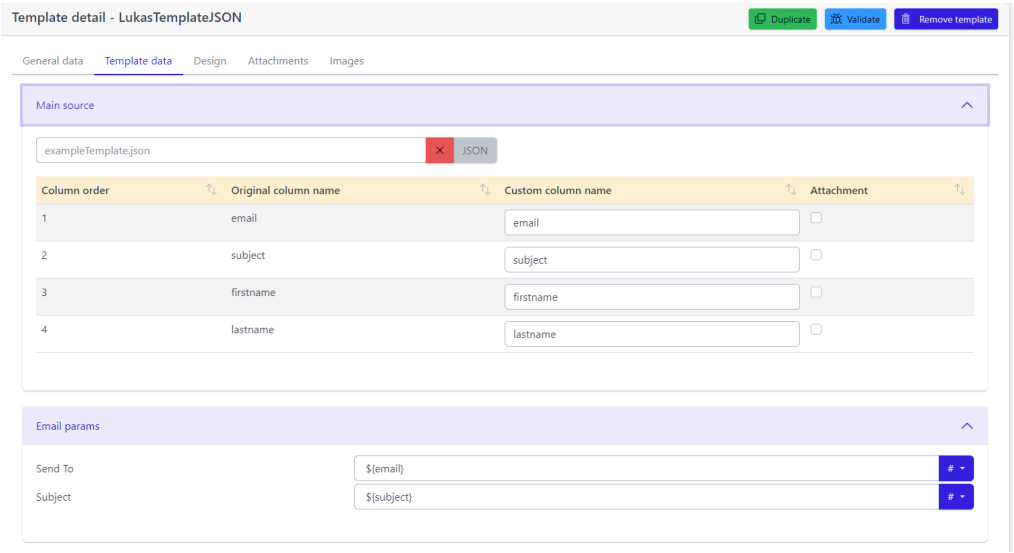

Figure 27: Example for EmailSendWithTemplateAdv - template data

In the following figure 28, we can see example of designed of email with used variables from JSON file.

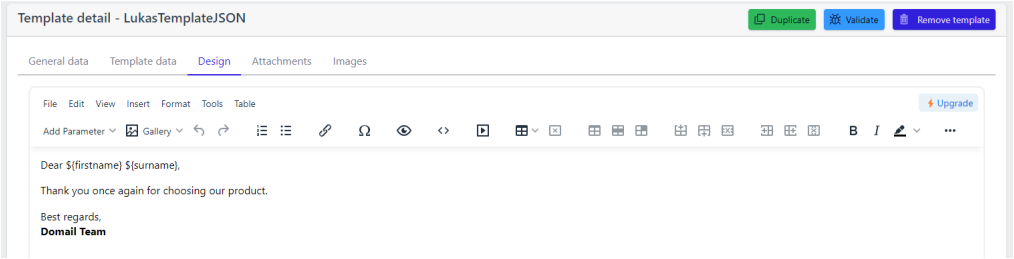

Figure 28: Example for EmailSendWithTemplateAdv - design

In the following listing 16, we can see JSON data for template.

```
   [{ "email": "lukas.vanek+soap-template1@dominanz.sk",
  "subject": "Subject of email 1",
  "firstname" : "Lukas",
  "surname" : "V."},
  {"email": "lukas.vanek+soap-template2@dominanz.sk",
  "subject": "Subject of email 2",
  "firstname" : "Lukas",
  "surname" : "V."},
  {"email": "lukas.vanek+soap-template3@dominanz.sk",
  "subject": "Subject of email 3",
  "firstname" : "Lukas",
  "surname" : "V."},
  {"email": "lukas.vanek+soap-template4@dominanz.sk",
  "subject": "Subject of email 4",
  "firstname" : "Lukas",
  "surname" : "V."}
]
```

Listing 16: JSON data for example for EmailSendWithTemplateAdv

In the following listing 17, we can see SOAP XML request for EmailSendWithTemplateAdv. In the element JSONData is Base64 encoded next JSON 16 :

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:p1="
    urn:dominanz.sk/domail">
    <soap:Header></soap:Header>
    <soap:Body>
    <p1:emailSendWithTemplateAdvRequest>
        <scenario>LukasTemplateJSON</scenario>
        <sysId>34242</sysId>
        <extId>1113</extId>
        <templateDataJSON>
            <JSONData>
    W3sgImVtYWlsIjogImx1a2FzLnZhbmVrK3NvYXAtdGVtcGxhdGUxQGRvbWluYW56LnNrIiwN
    Cgkic3ViamVjdCI6ICJTdWJqZWN0IG9mIGVtYWlsIDEiLA0KCSJmaXJzdG5hbWUiIDogIkx1a2FzIiwNCg
    kic3VybmFtZSIgOiAiVi4ifSwNCgl7ImVtYWlsIjogImx1a2FzLnZhbmVrK3NvYXAtdGVtcGxhdGUyQGRv
    bWluYW56LnNrIiwNCgkic3ViamVjdCI6ICJTdWJqZWN0IG9mIGVtYWlsIDIiLA0KCSJmaXJzdG5hbWUiID
    ogIkx1a2FzIiwNCgkic3VybmFtZSIgOiAiVi4ifSwNCgl7ImVtYWlsIjogImx1a2FzLnZhbmVrK3NvYXAt
    dGVtcGxhdGUzQGRvbWluYW56LnNrIiwNCgkic3ViamVjdCI6ICJTdWJqZWN0IG9mIGVtYWlsIDMiLA0KCS
    JmaXJzdG5hbWUiIDogIkx1a2FzIiwNCgkic3VybmFtZSIgOiAiVi4ifSwNCgl7ImVtYWlsIjogImx1a2Fz
    LnZhbmVrK3NvYXAtdGVtcGxhdGU0QGRvbWluYW56LnNrIiwNCgkic3ViamVjdCI6ICJTdWJqZWN0IG9mIG
            VtYWlsIDQiLA0KCSJmaXJzdG5hbWUiIDogIkx1a2FzIiwNCgkic3VybmFtZSIgOiAiVi4ifQ0KXQ
    ==
```

```
            </JSONData>
        </templateDataJSON>
    </p1:emailSendWithTemplateAdvRequest>
    </soap:Body>
</soap:Envelope>
```

Listing 17: SOAP request for emailSendWithTemplateAdv

After the request is sent, a SOAP response is returned 18.

```
    <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
        <SOAP-ENV:Header/>
        <SOAP-ENV:Body>
            <ns3:emailSendWithTemplateResponse xmlns:ns3="urn:dominanz.sk/domail">
                <metaCommunication>1747</metaCommunication>
                <subCommunication>1748</subCommunication>
                <subCommunication>1749</subCommunication>
                <subCommunication>1750</subCommunication>
                <subCommunication>1751</subCommunication>
            </ns3:emailSendWithTemplateResponse>
        </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
```

Listing 18: SOAP response for emailSendWithTemplateAdv

Response contains metaCommunication ID and 4 subCommunications and their IDs. In the following figure 29, we can see part of list of communications.

Id	Address	Processing	Delivery	Scenario	Group	Category	Send time
1751	lukas.vanek+soap-template4@dominanz.sk			LukasTemplateJSON			08.08.2023 13:03:47
1750	lukas.vanek+soap-template3@dominanz.sk			LukasTemplateJSON			08.08.2023 13:03:47
1749	lukas.vanek+soap-template2@dominanz.sk			LukasTemplateJSON			08.08.2023 13:03:46
1748	lukas.vanek+soap-template1@dominanz.sk			LukasTemplateJSON			08.08.2023 13:03:46
1747				LukasTemplateJSON			08.08.2023 13:03:46

Figure 29: List of communications for SOAP Request for EmailSendWithTemplate

3 REST API

The definition of the REST API is written in the file YAML in Openapi version: 3.0.2, which can be downloaded from this Domail website, which shows the definition in Swagger:
http://{domail_address}/swg/swagger-ui.html

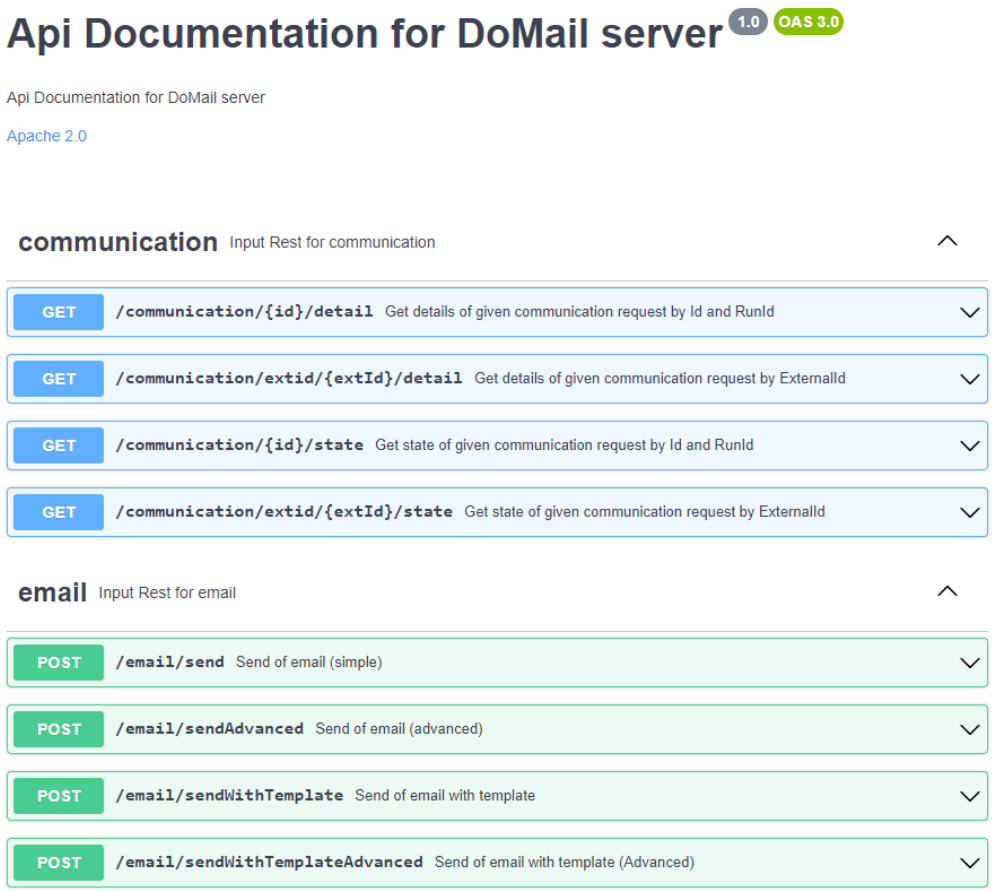

Figure 30: REST API Documentation for DoMail server

3.1 Definition calls

3.1.1 Communication

The following endpoints are available in communication:

- GET /communication/{id}/detail
- GET /communication/extid/{extId}/detail
- GET /communication/{id}/state
- GET /communication/extid/{extId}/detail

3.1.1.1 /communication/{id}/detail

- HTTP GET request
- Get details of given communication request by Id
- The input parameter is the id. This is the return value when a request is successfully sent to the Domail system.
- Response is JSON object type of 'CommunicationDetail'. The model of the 'CommunicationDetail' object is shown in the following 2 figures 31 and 32.

```
communicationDetail ⌄ {
    id*                     integer($int64)
    rcvTime                 string($date-time)
    sysId                   std50Regex  string
                            pattern: [-0-9a-zA-Z_@#/]+
                            maxLength: 50
    extId                   std128Regex  string
                            pattern: [-0-9a-zA-Z_@#/]+
                            maxLength: 128
    lastProcessingState     processingStateType  string
                            Enum:
                             ⌄ [ RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED ]
    lastDeliveryState       deliveryStateType  string
                            Enum:
                             ⌄ [ DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS ]
    scenario                std255Regex  string
                            pattern: [-0-9a-zA-Z_@#/]+
                            maxLength: 255
    testmode                boolean
                            default: false
    statistics              statisticsType ⌄ {
                                group                   std50Regex  string
                                                        pattern: [-0-9a-zA-Z_@#/]+
                                                        maxLength: 50
                                operation               std50Regex  string
                                                        pattern: [-0-9a-zA-Z_@#/]+
                                                        maxLength: 50
                                category                std50RegexOpt  string
                                                        maxLength: 50
                                                        pattern: [-0-9a-zA-Z_@#/]*
                                tags                     ⌄ [
                                                        maxItems: 5
                                                        std50Regex  string
                                                        pattern: [-0-9a-zA-Z_@#/]+
                                                        maxLength: 50]
                            }
```

Figure 31: Model of object CommunicationDetail 1

```
advancedRequestData   advancedRequestType ⌄ {
                        priority            boolean
                        duplicity           boolean
                        sendTime            string($date-time)

                      }
runs                   ⌄ [communicationProcDataType ⌄ {
                        runId               integer
                        procTime            string($date-time)
                        scenarioName        string255  string
                                            maxLength: 255
                        processingState     processingStateType  string
                                            Enum:
                                             ⌄ [ RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR,
                                            CANCELLED ]
                        deliveryState       deliveryStateType  string
                                            Enum:
                                             ⌄ [ DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS ]
                        sentEmails           ⌄ [sendCommunicationType ⌄ {
                                              addressNumber*      integer
                                              addressName         addressName  string
                                                                  maxLength: 200
                                              emailAddress        emailAddress  string
                                                                  maxLength: 512
                                                                  pattern: [^@]+@[^.]+..+
                                              processingState     processingStateType  string
                                                                  Enum:
                                                                   ⌄ [ RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING,
                                                                  DISPATCHED, DISPATCH_ERROR, CANCELLED ]
                                              deliveryState       deliveryStateType  string
                                                                  Enum:
                                                                   ⌄ [ DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED,
                                                                  DELIVERY_AMBIGUOUS ]
                                              messageId           string80  string
                                                                  maxLength: 80
                                              sendTime            string($date-time)

                                             }]

                       }]
}
```

Figure 32: Model of object CommunicationDetail 2

Object of type ‘CommunicationDetail’ has fields:

- **id** (required field)
  - main id of communication
- **rcvTime**
  - time of receipt of the communication
- **sysId**
  - the unique id/name of the external system wich use this the system Domail
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 50 characters
- **extId**
  - the unique id in external system wich use this the system Domail.
  - the Domail checks if duplicity is enabled/disabled. If ‘duplicity’ is disabled so it will not send more emails with the same ’extId’
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 128 characters
- **lastProcessingState**
  - last processing status

  - string
  - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}

- **lastDeliveryState**
  - last delivery status
  - string
  - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}

- **scenario**
  - the name of the script to be used for this communication
  - pattern: [-0-9a-zA-Z_@#/́]+
  - maximum length is 255 characters

- **testmode**
  - if the field is set to TRUE, the communication is in test mode
  - default value is FALSE

- **statistics**
  - statistical data that help to classify a given communication
  - **group**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **operation**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **category**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **tags**
    * maximum count of items is 5

- **advancedRequestData** - object type is 'AdvancedRequestType', subfields are:
  - **priority**
    * if we want to send a high priority email, we set the value to TRUE
    * boolean
    * value is true/false
  - **duplicity**
    * if we want to send more emails with the same extId, we have to set the value to TRUE

    - boolean
    - value is true/false
  - **sendTime**
    - If it is set, the email will be sent at that time
    - string (date-time)

- **runs** - object type is 'CommunicationProcDataType', subfields are:
  - **runId**
    - running id of communication
    - integer
    - value have to be greater as 0
  - **procTime**
    - date and time of processing
    - string (date-time)
  - **scenarioName**
    - the name of the script was used for this communication
    - maximum length is 255 characters
  - **processingState**
    - processing status
    - string
    - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE,
      PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
  - **deliveryState**
    - last delivery status
    - string
    - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
  - **sentEmails**
    - more emails can be generated during the processing of the communication, here are the details of each
    - object type is 'SendCommunicationType'
    - subfields are:
    - **addressNumber** (required field)
      - email number in communication processing
      - integer
    - **emailAddress**
      - email address
      - pattern: `[^@]+@[^.]+..+`
      - maximum length is 512 characters
    - **processingState**

    - processing status
    - string
    - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
  - **deliveryState**
    - last delivery status
    - string
    - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
  - **messageId**
    - message ID
    - maximum length is 80 characters
  - **sendTime**
    - If it is set, the email will be sent at that time
    - string (date-time)

3.1.1.2 /communication/extid/{extId}/detail

- HTTP GET request
- Get details of given communication request by ExternalId
- The input parameter is the ExternalId - extId. ExternalId is the ID provided to the request when calling theMail system.
- Response is also JSON object type of 'CommunicationDetail'. The model of the 'CommunicationDetail' object is shown in the figures 31 and 32.

3.1.1.3 /communication/{id}/state

- HTTP GET request
- Get state of given communication request by Id
- The input parameter is the id. This is the return value when a request is successfully sent to the system DoMail .

In the following figure 33 is the response JSON object of type Ćommunication-StateType:

```
communicationStateType ⌄ {
    lastProcessingState     processingStateType  string
                            Enum:
                             ⌄ [ RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED,
                            DISPATCH_ERROR, CANCELLED ]
    lastDeliveryState       deliveryStateType  string
                            Enum:
                             ⌄ [ DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS ]
    id*                     integer($int64)
    runs                     ⌄ [StateRunType ⌄ {
                                processingState        processingStateType  string
                                                       Enum:
                                                        ⌄ [ RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED,
                                                       DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED ]
                                deliveryState          deliveryStateType  string
                                                       Enum:
                                                        ⌄ [ DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN,
                                                       DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS ]
                                runId                  integer($int32)
                                sentEmails              ⌄ [StateEmailType ⌄ {
                                                          addressNumber         integer
                                                          addressName           addressName  string
                                                                                maxLength: 200
                                                          emailAddress          emailAddress  string
                                                                                maxLength: 512
                                                                                pattern: [^@]+@[^.]+..+
                                                          processingState       processingStateType  string
                                                                                Enum:
                                                                                 > Array [ 10 ]
                                                          deliveryState         deliveryStateType  string
                                                                                Enum:
                                                                                 > Array [ 5 ]

                                                         }]

                             }]

}
```

Figure 33: Model of object CommunicationStateType

Object of type ’CommunicationStateType’ has fields:

- **lastProcessingState**
  - last processing status
  - string
  - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
- **lastDeliveryState**
  - last delivery status
  - string
  - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
- **id** (required field)
  - main id of communication

- **runs** - object type is 'CommunicationProcDataType', subfields are:
  - **runId**
    - running id of communication
    - integer
    - value have to be greater as 0
  - **procTime**
    - date and time of processing
    - string (date-time)
  - **scenarioName**
    - the name of the script was used for this communication
    - maximum length is 255 characters
  - **processingState**
    - processing status
    - string
    - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE,
      PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
  - **deliveryState**
    - last delivery status
    - string
    - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
  - **sentEmails**
    - more emails can be generated during the processing of the communication, here are the details of each
    - object type is 'SendCommunicationType'
    - subfields are:
    - **addressNumber** (required field)
      - email number in communication processing
      - integer
    - **emailAddress**
      - email address
      - pattern: `[^@]+@[^.]+..+`
      - maximum length is 512 characters
    - **processingState**
      - processing status
      - string
      - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
    - **deliveryState**

        - last delivery status
        - string
        - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
    - **messageId**
        - message ID
        - maximum length is 80 characters
    - **sendTime**
        - If it is set, the email will be sent at that time
        - string (date-time)

3.1.1.4 /communication/extid/{extId}/state

- HTTP GET request
- Get state of given communication request by ExternalId
- The input parameter is the ExternalId - extId. ExternalId is the ID provided to the request when calling theMail system.
- Response is also JSON object type of 'CommunicationStateType'. The model of the 'CommunicationStateType' object is shown in the figure 33.

3.1.2 Email

The following endpoints are available in email:

- POST /email/send
- POST /email/sendAdvanced
- POST /email/sendWithTemplate
- POST /email/sendWithTemplateAdvanced

3.1.2.1 /email/send

- HTTP POST request,
- Endpoint is used to simply send an email via REST API,
- Body of POST request is JSON object type of 'emailSendRequest',
- Response is object type of 'CommunicationId'

In the following figure 34 is the definition of the JSON object 'emailSendRequest':

```
emailSendRequest {
   to*                    [
                          maxItems: 100
                          emailAddressType {
                             addressName          addressName  string
                                                  maxLength: 200
                             emailAddress*        emailAddress  string
                                                  maxLength: 512
                                                  pattern: [^@]+@[^.]+..+
                           }]
   subject                string255  string
                          maxLength: 255
   htmlBody               string
   plainTextBody          string
   scenario*              std255Regex  string
                          pattern: [-0-9a-zA-Z_@#/]+
                          maxLength: 255
   testmode               boolean
                          default: false
   attachments            [
                          maxItems: 20
                          fileHandlerType {
                             filename*            string255  string
                                                  maxLength: 255
                             contentID            string
                             mimeType             string
                             encoding             string
                             dataHandler*         string($byte)
                                                  pattern: ^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}==|[A-Za-z0-9+/]{3}=)?$
                                                  maxLength: 30720000
                                                  xml: OrderedMap { "name": "dataHandler", "attribute": false, "wrapped": false }
                                                  xml:
                                                     name: dataHandler
                                                     attribute: false
                                                     wrapped: false
                           }]
   statistics             statisticsType {
                             group                std50Regex  string
                                                  pattern: [-0-9a-zA-Z_@#/]+
                                                  maxLength: 50
                             operation            std50Regex  string
                                                  pattern: [-0-9a-zA-Z_@#/]+
                                                  maxLength: 50
                             category             std50RegexOpt  string
                                                  maxLength: 50
                                                  pattern: [-0-9a-zA-Z_@#/]*
                             tags                 [
                                                  maxItems: 5
                                                  std50Regex  string
                                                  pattern: [-0-9a-zA-Z_@#/]+
                                                  maxLength: 50]
                          }
}
```

Figure 34: Object type of ’emailSendRequest’

- **to** (required field)
  - list of email address of recipients
  - minimum count of recipients is 1 and maximum 100 recipients
- **subject**
  - subject of email
  - maximum length is 255 characters
- **htmlBody**
  - html text body of email
  - string
- **plainTextBody**
  - plain text body of email
  - string
- **scenario** (required field)
  - the name of the script to be used for this communication

  - pattern: [-0-9a-zA-Z_@#/́]+
  - maximum length is 255 characters

- **testmode**
  - if the field is set to TRUE, the communication is in test mode
  - default value is FALSE

- **attachments**
  - array of attachments
  - maximum count of items is 20
  - One item of type **fileHandlerType**:
    - **filename** (required field)
      - file name
      - maximum length is 255 characters
    - **contentId**
      - Content ID
      - string
    - **mimeType**
      - MimeType (Example: application/text)
      - string
    - **encoding**
      - Encoding (Example: utf-8)
      - string
    - **dataHandler** (required field)
      - Base64 string

- **statistics**
  - statistical data that help to classify a given communication
  - **group**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **operation**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **category**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **tags**
    - maximum count of items is 5

In the following figure 35 is the definition of the JSON object ’CommunicationId’:

```
communicationId ⌄ {
    id*                    integer($int64)

}
```

Figure 35: Object type of ’CommunicationId’

Response JSON object ’CommunicationId’:

- **id**
  - Id of send communication in the system Domail,
  - It has to be bigger than 0

3.1.2.2 /email/sendAdvanced

- HTTP POST request,
- Send email (advanced),
- Body of POST request is JSON object type of ’EmailSendAdvancedRequest’,
- Response is object type of ’CommunicationId’

Object type of ’EmailSendAdvancedRequest’ (36) is inherited from object type of ’EmailSendRequest’. ’EmailSendAdvancedRequest’ contains all fields from object type ’EmailSendAdvancedRequest’ (green part of from picture 36).

```
emailSendRequest {
  to*                    [
                         maxItems: 100
                         emailAddressType {
                           addressName          addressName  string
                                                maxLength: 200
                           emailAddress*        emailAddress  string
                                                maxLength: 512
                                                pattern: [^@]+@[^.]+..+
                         }]
  subject                string255  string
                         maxLength: 255
  htmlBody               string
  plainTextBody          string
  scenario*              std255Regex  string
                         pattern: [-0-9a-zA-Z_@#/]+
                         maxLength: 255
  testmode               boolean
                         default: false
  attachments            [
                         maxItems: 20
                         fileHandlerType {
                           filename*            string255  string
                                                maxLength: 255
                           contentID            string
                           mimeType             string
                           encoding             string
                           dataHandler*         string($byte)
                                                pattern: ^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}==|[A-Za-z0-9+/]{3}=)?$
                                                maxLength: 30720000
                                                xml: OrderedMap { "name": "dataHandler", "attribute": false, "wrapped": false }
                                                xml:
                                                  name: dataHandler
                                                  attribute: false
                                                  wrapped: false
                         }]
  statistics             statisticsType {
                           group                std50Regex  string
                                                pattern: [-0-9a-zA-Z_@#/]+
                                                maxLength: 50
                           operation            std50Regex  string
                                                pattern: [-0-9a-zA-Z_@#/]+
                                                maxLength: 50
                           category             std50RegexOpt  string
                                                maxLength: 50
                                                pattern: [-0-9a-zA-Z_@#/]*
                           tags                 [
                                                maxItems: 5
                                                std50Regex  string
                                                pattern: [-0-9a-zA-Z_@#/]+
                                                maxLength: 50]
                         }
  sysId*                 std50Regex  string
                         pattern: [-0-9a-zA-Z_@#/]+
                         maxLength: 50
  extId*                 std128Regex  string
                         pattern: [-0-9a-zA-Z_@#/]+
                         maxLength: 128
  advancedRequestData    advancedRequestType {
                           priority             boolean
                           duplicity            boolean
                           sendTime             string($date-time)
                         }
  additionalContent      [additionalContentType {
                           key                  std50Regex  string
                                                pattern: [-0-9a-zA-Z_@#/]+
                                                maxLength: 50
                           value                {
                                                }
                         }]
}
```

Figure 36: Object type of ’EmailSendAdvancedRequest’

New fields (red part of from picture 36) are:

- **sysId**
  - the unique id/name of the external system wich use this the system Domail
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 50 characters
- **extId**
  - the unique id in external system wich use this the system Domail.

  - the Domail checks if duplicity is enabled/disabled. If 'duplicity' is disabled so it will not send more emails with the same 'extId'
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 128 characters

- **advancedRequestData** - object type is 'AdvancedRequestType', subfields are:
  - **priority**
    * if we want to send a high priority email, we set the value to TRUE
    * boolean
    * value is true/false
  - **duplicity**
    * if we want to send more emails with the same extId, we have to set the value to TRUE
    * boolean
    * value is true/false
  - **sendTime**
    * If it is set, the email will be sent at that time
    * string (date-time)
    * value have to be time greater than now
- **additionalContent** - array of objects type 'additionalContentType'. Subfields of object type 'additionalContentType' are:
  - **key**
    * Key is unique name of item
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **value**
    * Value of item
    * object

3.1.2.3 /email/sendWithTemplate

- HTTP POST request
- Send of email with template
- Body of POST request is JSON object type of 'EmailSendWithTemplateRequest',
- Response is object type of 'CommunicationTemplateId'

In the following figure 37 is the definition of the JSON object 'EmailSendWithTemplateRequest':

```
emailSendWithTemplateRequest ⌄ {
   scenario*            std255Regex  string
                        pattern: [-0-9a-zA-Z_@#/]+
                        maxLength: 255
   statistics           statisticsType ⌄ {
                           group                 std50Regex  string
                                                 pattern: [-0-9a-zA-Z_@#/]+
                                                 maxLength: 50
                           operation             std50Regex  string
                                                 pattern: [-0-9a-zA-Z_@#/]+
                                                 maxLength: 50
                           category              std50RegexOpt  string
                                                 maxLength: 50
                                                 pattern: [-0-9a-zA-Z_@#/]*
                           tags
                                                  ⌄ [
                                                 maxItems: 5
                                                 std50Regex  string
                                                 pattern: [-0-9a-zA-Z_@#/]+
                                                 maxLength: 50]
                        }
   testmode             boolean
                        default: false
   params                ⌄ [parametersTemplate ⌄ {
                           key                   std50Regex  string
                                                 pattern: [-0-9a-zA-Z_@#/]+
                                                 maxLength: 50
                           value                 string1024  string
                                                 maxLength: 1024
                         }]
   attachments           ⌄ [
                        maxItems: 20
                        fileHandlerType ⌄ {
                           filename*             string255  string
                                                 maxLength: 255
                           contentID             string
                           mimeType              string
                           encoding              string
                           dataHandler*          string($byte)
                                                 pattern: ^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}==|[A-Za-
                                                 z0-9+/]{3}=)?$
                                                 maxLength: 30720000
                                                 xml: OrderedMap { "name": "dataHandler", "attribute": false,
                                                 "wrapped": false }
                                                 xml:
                                                    name: dataHandler
                                                    attribute: false
                                                    wrapped: false
                         }]
}
```

Figure 37: Object type of ’EmailSendWithTemplateRequest’

Properties of object type ’EmailSendWithTemplateRequest’:

- **scenario** (required field)
  - the name of the script to be used for this communication
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 255 characters
- **statistics** - JSON object type of ’StatisticsType’
  - statistical data that help to classify a given communication
  - **group**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **operation**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters

  - **category**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **tags**
    - maximum count of items is 5
    - pattern: [-0-9a-zA-Z_@#/]+

- **testmode**
  - if the field is set to TRUE, the communication will be to send in test mode
  - default value is FALSE

- **params**
  - list of parameters to be used in the template
  - array of objects type of 'ParametersTemplate'
  - **key**
    - string
    - pattern: [-0-9a-zA-Z_@#/]+
  - **value**
    - string
    - maximum length is 1024 characters

- **attachments**
  - array of attachments
  - maximum count of items is 20
  - One item of type **fileHandlerType**:
    - **filename** (required field)
      - file name
      - maximum length is 255 characters
    - **contentId**
      - Content ID
      - string
    - **mimeType**
      - MimeType (Example: application/text)
      - string
    - **encoding**
      - Encoding (Example: utf-8)
      - string
    - **dataHandler** (required field)
      - Base64 string

In the following figure 38 is the definition of the JSON object 'CommunicationTemplateId':

```
communicationTemplateId ⌄ {
    mainCommunicationId  integer($int64)
    subCommunicationId
                          ⌄ [integer($int64)]
}
```

Figure 38: Object type of 'CommunicationTemplateId'

Response JSON object 'CommunicationTemplateId':

- **mainCommunicationId**
  - id of the main meta communication,
  - It has to be bigger than 0
- **subCommunicationId** - array of type integer 64bit (long)
  - List of subcommunication IDs that were created based on the defined template

Now follows the HTTP POST example (19) to call endpoint:
http:\\{yourUrl}/dir/domail-input-rest/email/sendWithTemplate

```
{
    "scenario": "testTemplate"
}
```

Listing 19: Example for POST request for sending email with template only with required fields

After successful sending, the response will arrive as it is in 20.

```
{
    "mainCommunicationId": 1438,
    "subCommunicationId": []
}
```

Listing 20: Example for POST request for sending email with template only with required fields

3.1.2.4 /email/sendWithTemplateAdvanced

- HTTP POST request
- Send of email with template (Advanced)
- Body of POST request is JSON object type of 'EmailSendWithTemplateAdvancedRequest',
- Response is object type of 'CommunicationTemplateId'

In the following figure 39 is the definition of the JSON object ’EmailSendWithTemplateAdvancedRequest’:

```
emailSendWithTemplateAdvancedRequest ⌄ {
    scenario*           std255Regex  string
                        pattern: [-0-9a-zA-Z_@#/]+
                        maxLength: 255
    sysId*              std50Regex  string
                        pattern: [-0-9a-zA-Z_@#/]+
                        maxLength: 50
    extId*              std128Regex  string
                        pattern: [-0-9a-zA-Z_@#/]+
                        maxLength: 128
    statistics          statisticsType ⌄ {
                            group               std50Regex  string
                                                pattern: [-0-9a-zA-Z_@#/]+
                                                maxLength: 50
                            operation           std50Regex  string
                                                pattern: [-0-9a-zA-Z_@#/]+
                                                maxLength: 50
                            category            std50RegexOpt  string
                                                maxLength: 50
                                                pattern: [-0-9a-zA-Z_@#/]*
                            tags                ⌄ [
                                                maxItems: 5
                                                std50Regex  string
                                                pattern: [-0-9a-zA-Z_@#/]+
                                                maxLength: 50]
                        }
    testmode            boolean
                        default: false
    advancedRequestData advancedRequestType ⌄ {
                            priority            boolean
                            duplicity           boolean
                            sendTime            string($date-time)
                        }
    attachments          ⌄ [
                        maxItems: 50000
                        fileHandlerType ⌄ {
                            filename*           string255 › [...]
                            contentID           string
                            mimeType            string
                            encoding            string
                            dataHandler*        string($byte)
                                                pattern: ^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}==|[A-Za-z0-9+/]
                                                {3}=)?$
                                                maxLength: 30720000
                                                xml: OrderedMap { "name": "dataHandler", "attribute": false,
                                                "wrapped": false }
                                                xml:
                                                    name: dataHandler
                                                    attribute: false
                                                    wrapped: false
                         }]
    templateDataJSON    templateDataJSON ⌄ [ ⌄ {
                         }]
    templateDataCSV     templateDataCSV ⌄ [fileHandlerType › {...}]
}
```

Figure 39: Object type of ’EmailSendWithTemplateAdvancedRequest’

Properties of object type ’EmailSendWithTemplateAdvancedRequest’:

- **scenario** (required field)
  - the name of the script to be used for this communication
  - pattern: [-0-9a-zA-Z_@#/́]+
  - maximum length is 255 characters
- **sysId** (required field)
  - the unique id/name of the external system wich use this the system Domail
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 50 characters

- **extId** (required field)
  - the unique id in external system wich use this the system Domail.
  - the Domail checks if duplicity is enabled/disabled. If 'duplicity' is disabled so it will not send more emails with the same 'extId'
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 128 characters
- **statistics** - JSON object type of 'StatisticsType'
  - statistical data that help to classify a given communication
  - **group**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **operation**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **category**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **tags**
    * maximum count of items is 5
    * pattern: [-0-9a-zA-Z_@#/]+
- **testmode**
  - if the field is set to TRUE, the communication will be to send in test mode
  - default value is FALSE
- **advancedRequestData** - object type is 'AdvancedRequestType', subfields are:
  - **priority**
    * if we want to send a high priority email, we set the value to TRUE
    * boolean
    * value is true/false
  - **duplicity**
    * if we want to send more emails with the same extId, we have to set the value to TRUE
    * boolean
    * value is true/false
  - **sendTime**
    * If it is set, the email will be sent at that time
    * string (date-time)
    * value have to be time greater than now
- **attachments**

  - array of attachments
  - maximum count of items is 20
  - One item of type **fileHandlerType**:
    * **filename** (required field)
      · file name
      · maximum length is 255 characters
    * **contentId**
      · Content ID
      · string
    * **mimeType**
      · MimeType (Example: application/text)
      · string
    * **encoding**
      · Encoding (Example: utf-8)
      · string
    * **dataHandler** (required field)
      · Base64 string

- **templateDataJSON**
  - array of any objects
- **templateDataCSV**
  - object of type **fileHandlerType** - same as was in attachments:
    * **filename** (required field)
      · file name
      · maximum length is 255 characters
    * **contentId**
      · Content ID
      · string
    * **mimeType**
      · MimeType (Example: application/text)
      · string
    * **encoding**
      · Encoding (Example: utf-8)
      · string
    * **dataHandler** (required field)
      · Base64 string

3.2 Create OpenAPI client in C#

In the following example 21 there is an object of type CustomOpenApiClient, which is inherited from the generated object of type 'Client'. The generated object 'Client' was created based on the definition written in the YAML file that describes the REST API. The inherited object 'CustomOpenApiClient' is needed to change the settings

in ’JsonSerializerSettings.DateFormatString’ to "yyyyMMddTHH:mm:ss.fffffffZ" and ’JsonSerializerSettings.DateTimeZoneHandling’ to Newtonsoft.Json.DateTimeZoneHandling.Local, because the system Domail works with the local time of the server.

```
public class CustomOpenApiClient : Client
{
    public CustomOpenApiClient(string baseUrl) : base(baseUrl, new HttpClient())
    {
        this.UpdateSerializerSettings();
    }

    private void UpdateSerializerSettings()
    {
        //2023-04-14T14:39:55.9794317
        base.JsonSerializerSettings.DateFormatString = "yyyy-MM-ddTHH:mm:ss.fffffffZ";

        //base.JsonSerializerSettings.DateFormatHandling = Newtonsoft.Json.
    DateFormatHandling.MicrosoftDateFormat;
        base.JsonSerializerSettings.DateTimeZoneHandling = Newtonsoft.Json.
    DateTimeZoneHandling.Local;

    }
}
```

Listing 21: Object type of ’CustomOpenApiClient’ inherited from ’Client’

4 Change state notification queues

- ake MQ existuju, popisat, dopisat co sa v nich nachadza... ? -priklady na pouzitie a spracovanie - doplnit priklad spracovania notifikacii z MQ

5 Scripts

In these examples we will show how different programming languages can be used to send an email (create a communication) or to get details about the requested communication.

5.1 Example 1 - Send email - EmailSendSimple

5.1.1 through REST API

In this part it is a REST API call.

5.1.1.1 JSON

Now follows a simple HTTP POST example to call endpoint:
http:\\{yourUrl}/dir/domail-input-rest/email/send

We can test this e.g. via Postman.

```
{
    "to": [
        {
          "addressName": "NameOfEmail",
          "emailAddress": "email@email.com"
        }
    ]
}
```

Listing 22: Example for POST request for sending email (simple) only with required fields

5.1.1.2 C#

Another language is C#, in which we used an object of type 'CustomOpenApiClient' created by us, which is inherited from the object of type 'Client'.

At the beginning we need to initialize an instance of the 'CustomOpenApiClient' object, through which we connect to the doMail system.

```
//Create CustomOpenApiClient
string url = $"{Configuration.DOMAIL_URL}/dir/domail-input-rest/";
var client = new CustomOpenApiClient(url);
```

Next, we will prepare an object of type 'EmailSendRequest', which will contain all the information about the request sent to the server. We fill the same fields as we filled in the example in JSON.

```
//Create EmailSendRequest
EmailSendRequest emailRequest = new EmailSendRequest();

//set 1 recipients
emailRequest.To = new List<EmailAddressType>() {
        new EmailAddressType() { AddressName = "NameOfEmail", EmailAddress = "
    email@email.com"}
};
```

Now the prepared request is sent via the ASYNC call provided by the system doMail. Since we want to wait for a response, we wait for processing to print the return.

```
//send reqest throug REST API EmailSendSimpleAsync
Task<CommunicationId> taskEmailSendSimpleAsync = client.EmailSendSimpleAsync(
    emailRequest);

//wait for resposne
var result = taskEmailSendSimpleAsync.GetAwaiter().GetResult();

if (result.Id > 0)
    Console.WriteLine($"call POST OK - result.Id={result.Id}");
```

```
try
{
    ...
}
catch (ApiException<RestErrorResponse> e1)
{
    Console.WriteLine($"ApiException: {e1}\nStatusCode: {e1.StatusCode}\nResult.
    ResponseCode: {e1.Result?.ResponseCode}\nResult.Message: {e1.Result?.Message}");
}
catch (ApiException e2)
{
    Console.WriteLine($"ApiException: {e2}, StatusCode: {e2.StatusCode}");
}
catch (Exception ex)
{
    Console.WriteLine($"Exception: {ex}");
}
```

Complete source file is on the next listening 23 or the whole solution with examples can be downloaded from: TODO: WWW

```
try
{
    //Create CustomOpenApiClient
    string url = $"{Configuration.DOMAIL_URL}/dir/domail-input-rest/";
    var client = new CustomOpenApiClient(url);

    //Create EmailSendRequest
    EmailSendRequest emailRequest = new EmailSendRequest();

    //set 1 recipients
    emailRequest.To = new List<EmailAddressType>() {
            new EmailAddressType() { AddressName = "NameOfEmail", EmailAddress = "
    email@email.com"}
    };

    //send reqest throug REST API EmailSendSimpleAsync
    Task<CommunicationId> taskEmailSendSimpleAsync = client.EmailSendSimpleAsync(
    emailRequest);

    //wait for resposne
    var result = taskEmailSendSimpleAsync.GetAwaiter().GetResult();

    if (result.Id > 0)
        Console.WriteLine($"call POST OK - result.Id={result.Id}");
}
catch (ApiException<RestErrorResponse> e1)
{
    Console.WriteLine($"ApiException: {e1}\nStatusCode: {e1.StatusCode}\nResult.
    ResponseCode: {e1.Result?.ResponseCode}\nResult.Message: {e1.Result?.Message}");
}
```

```
catch (ApiException e2)
{
    Console.WriteLine($"ApiException: {e2}, StatusCode: {e2.StatusCode}");
}
catch (Exception ex)
{
    Console.WriteLine($"Exception: {ex}");
}
```

Listing 23: Example for POST request for sending email (simple) only with required fields in C#

5.1.1.3 Java

TODO

5.1.1.4 Python

TODO

5.1.2 through SOAP

In this part it is a SOAP call.

5.1.2.1 XML Request

Now follows a simple HTTP SOAP XML request example to call endpoint:
We can test this e.g. via Postman.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:p1="
     urn:dominanz.sk/domail">
    <soap:Header>
    </soap:Header>
    <soap:Body>
        <p1:emailSendRequestElement>
        <to>
            <name>NameOfEmail</name>
            <emailAddress>lukas.vanek+soap@dominanz.sk</emailAddress>
        </to>
        <subject>Test SOAP email</subject>
        <scenario>Lukas-SOAP/REST</scenario>
    </p1:emailSendRequestElement>
    </soap:Body>
</soap:Envelope>
```

Listing 24: Example for POST request for sending email with required fields

5.1.2.2 C#

```
//Create Client
string url = $"{Configuration.DOMAIL_URL}/dws/domail-input-ws/";
var client = new dominanz_domail_Communication();
client.Url = url;
```

Next, we will prepare an object of type 'EmailSendRequest', which will contain all the information about the request sent to the server. We fill the same fields as we filled in the XML SOAP example.

```
//Create EmailSendRequest
EmailSendRequestType emailRequest = new EmailSendRequestType();
emailRequest.scenario = "Lukas-SOAP/REST";

//set 1 recipients
var listArray = new List<DoMailGeneratedLibrary.SOAP.EmailAddressType>() {
        new DoMailGeneratedLibrary.SOAP.EmailAddressType() {
            name = "NameOfEmail",
            emailAddress = "lukas.vanek+soap@dominanz.sk"}
};

emailRequest.to = listArray.ToArray();
```

```
//send reqeust through SOAP EmailSend
var result = client.emailSend(emailRequest);

if (result.id > 0)
    Console.WriteLine($"call POST OK - result.id={result.id}");
```

5.1.2.3 Java

TODO

5.1.2.4 Python

TODO

5.2 Example 2 - Send email - EmailSendSimple

The next example is an extension of the previous Example 1 , now we want to send an email with all fields

5.2.1 through REST API

In this part it is a REST API call.

5.2.1.1 JSON

In the next example 25, there is POST request with all fields:

```
{
    "to": [
     {
          "addressName": "NameOfEmail1",
          "emailAddress": "email1@email.com"
        },
     {
          "addressName": "NameOfEmail2",
          "emailAddress": "email2@email.com"
        }
  ],
  "attachments": [
    {
      "filename": "testFile1",
      "dataHandler": "VG90byBqZSB0ZXN0b3ZhY2lhIHByaWxvaGE=",
      "contentID": "54321",
```

```
      "mimeType": "application/text",
      "encoding": "utf-8"
    }
  ],
  "statistics": {
    "group": "testGroupName",
    "operation": "TI212",
    "category": "categoryName",
    "tags": [
      "tag1",
      "TEST"
    ]
  },
  "subject": "Subject of email",
  "htmlBody": "<b>Html body/b>",
  "scenario": "ScenarioName",
  "testmode": false
}
```

Listing 25: Example for POST request for sending email (simple) with all fields

5.2.1.2 C#

Unlike example 1, 2 recipients are set here and all parameters of the request are set on the pointer.

```
//Create EmailSendRequest
EmailSendRequest emailRequest = new EmailSendRequest();

//set 2 recipients
emailRequest.To = new List<EmailAddressType>() {
        new EmailAddressType() { AddressName = "NameOfEmail", EmailAddress = "email@email.com"},
        new EmailAddressType() { AddressName = "NameOfEmail2", EmailAddress = "email2@email.com"},

};

var utf8 = new UTF8Encoding();

emailRequest.Attachments = new List<FileHandlerType>(){
    new FileHandlerType()
    {
        Filename = "testFile1",
        DataHandler = utf8.GetBytes("VG90byBqZSB0ZXN0b3ZhY2lhIHByaWxvaGE"),
        ContentID = "54321",
        MimeType = "application/text",
        Encoding = "UTF-8"
    }
};

emailRequest.Statistics = new StatisticsType()
{
    Group = "testGroupName",
    Operation = "TI212",
    Category = "categoryName",
    Tags = new List<string>() { "tag1", "TEST" }
};
emailRequest.Subject = "Subject of email";
emailRequest.HtmlBody = "<b>Html body/b>";
emailRequest.Scenario = "ScenarioName";
emailRequest.Testmode = false;
```

Complete source file is on the next listening 26 or the whole solution with examples can be downloaded from: TODO: WWW

```
try
{
    //Create CustomOpenApiClient
    string url = $"{Configuration.DOMAIL_URL}/dir/domail-input-rest/";
    var client = new CustomOpenApiClient(url);

    //Create EmailSendRequest
    EmailSendRequest emailRequest = new EmailSendRequest();

    //set 2 recipients
    emailRequest.To = new List<EmailAddressType>() {
            new EmailAddressType() { AddressName = "NameOfEmail", EmailAddress = "email@email.com"},
            new EmailAddressType() { AddressName = "NameOfEmail2", EmailAddress = "email2@email.com"},

    };

    var utf8 = new UTF8Encoding();

    emailRequest.Attachments = new List<FileHandlerType>(){
        new FileHandlerType()
        {
            Filename = "testFile1",
            DataHandler = utf8.GetBytes("VG90byBqZSB0ZXN0b3ZhY2lhIHByaWxvaGE"),
            ContentID = "54321",
            MimeType = "application/text",
            Encoding = "UTF-8"
        }
    };

    emailRequest.Statistics = new StatisticsType()
    {
        Group = "testGroupName",
        Operation = "TI212",
        Category = "categoryName",
        Tags = new List<string>() { "tag1", "TEST" }
    };
    emailRequest.Subject = "Subject of email";
    emailRequest.HtmlBody = "<b>Html body/b>";
    emailRequest.Scenario = "ScenarioName";
    emailRequest.Testmode = false;

    //send reqest throug REST API EmailSendSimpleAsync
    Task<CommunicationId> taskEmailSendSimpleAsync = client.EmailSendSimpleAsync(emailRequest);

    //wait for resposne
    var result = taskEmailSendSimpleAsync.GetAwaiter().GetResult();

    if (result.Id > 0)
        Console.WriteLine($"call POST OK - result.Id={result.Id}");

}
catch (ApiException<RestErrorResponse> e1)
{
    Console.WriteLine($"ApiException: {e1}\nStatusCode: {e1.StatusCode}\nResult.ResponseCode: {e1.Result?.ResponseCode}\nResult.Message: {e1.Result?.Message}");
}
catch (ApiException e2)
```

```
{
    Console.WriteLine($"ApiException: {e2}, StatusCode: {e2.StatusCode}");
}
catch (Exception ex)
{
    Console.WriteLine($"Exception: {ex}");
}
```

Listing 26: Example for POST request for sending email (simple) only with required fields in C#

5.2.1.3 Java

TODO

5.2.1.4 Python

TODO

5.2.2 through SOAP

In this part it is a SOAP call.

5.2.2.1 XML Request

In the next example 27, there is POST request with all fields: We can test this e.g. via Postman.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:p1="
    urn:dominanz.sk/domail">
  <soap:Header>
  </soap:Header>
  <soap:Body>
      <p1:emailSendRequestElement>
        <to>
            <name>NameOfEmail1</name>
            <emailAddress>lukas.vanek+soap1@dominanz.sk</emailAddress>
        </to>
        <to>
            <name>NameOfEmail2</name>
            <emailAddress>lukas.vanek+soap2@dominanz.sk</emailAddress>
        </to>
        <subject>Subject of email from to SOAP request</subject>
        <htmlBody>Html body</htmlBody>
        <attachment>
            <p1:filename>testFile1</p1:filename>
            <p1:contentID>54321</p1:contentID>
            <p1:mimeType>application/text</p1:mimeType>
            <p1:encoding>utf-8</p1:encoding>
            <p1:dataHandler>{{attachmentText}}</p1:dataHandler>
        </attachment>
        <scenario>Lukas-SOAP/REST</scenario>
        <testMode>false</testMode>
        <statistics>
                <group>testGroupName</group>
                <operation>TI212</operation>
                <category>categoryName</category>
                <tag>tag1</tag>
```

```
                <tag>TEST</tag>
        </statistics>
    </p1:emailSendRequestElement>
  </soap:Body>
</soap:Envelope>
```

Listing 27: Example for POST request for sending email with required fields

5.2.2.2 C#

In this part it is a SOAP call, where we fill all the parameters.

```
//Create Client
string url = $"{Configuration.DOMAIL_URL}/dws/domail-input-ws/";
var client = new dominanz_domail_Communication();
client.Url = url;

//Create EmailSendRequest
EmailSendRequestType emailRequest = new EmailSendRequestType();
emailRequest.scenario = "Lukas-SOAP/REST";

//set 1 recipients
var listArray = new List<DoMailGeneratedLibrary.SOAP.EmailAddressType>() {
        new DoMailGeneratedLibrary.SOAP.EmailAddressType() {
            name = "NameOfEmail1",
            emailAddress = "lukas.vanek+soap1@dominanz.sk"},
        new DoMailGeneratedLibrary.SOAP.EmailAddressType() {
            name = "NameOfEmail2",
            emailAddress = "lukas.vanek+soap2@dominanz.sk"}
};
emailRequest.to = listArray.ToArray();

var utf8 = new UTF8Encoding();

var listAttachmentArray = new List<DoMailGeneratedLibrary.SOAP.FileHandlerType>()
{
    new DoMailGeneratedLibrary.SOAP.FileHandlerType()
    {
        filename = "testFile1",
        dataHandler = utf8.GetBytes("VG90byBqZSB0ZXN0b3ZhY2lhIHByaWxvaGE"),
        contentID = "54321",
        mimeType = "application/text",
        encoding = "UTF-8"
    }
};
emailRequest.attachment = listAttachmentArray.ToArray();

emailRequest.statistics = new DoMailGeneratedLibrary.SOAP.StatisticsType()
{
    group = "testGroupName",
    operation = "TI212",
    category = "categoryName",
    tag = new String[] { "tag1", "TEST" }
};

emailRequest.subject = "Subject of email from to SOAP request";
emailRequest.htmlBody = "<b>Html body/b>";
emailRequest.scenario = "Lukas-SOAP/REST";
emailRequest.testMode = false;

//send reqeust through SOAP EmailSend
var result = client.emailSend(emailRequest);
```

```

if (result.id > 0)
    Console.WriteLine($"call POST OK - result.id={result.id}");
```

5.2.2.3 Java

TODO

5.2.2.4 Python

TODO

5.3 Example 3 - Send email - EmailSendAdvanced

This example describes how to call EmailSendAdvanced only with required fields.

5.3.1 through REST API

In this part it is a REST API call.

5.3.1.1 JSON

Now follows the simple HTTP POST example (28) to call endpoint:
http:\\{yourUrl}/dir/domail-input-rest/email/sendAdvanced

```
{
  "extId": 100,
  "sysId": 200,
  "to": [
    {
          "addressName": "NameOfEmail",
          "emailAddress": "email@email.com"
    }
  ]
}
```

Listing 28: Example for POST request for sending email (advanced) only with required fields

5.3.1.2 C#

Complete source file is on the next listening 29 or the whole solution with examples can be downloaded from: TODO: WWW

```
try
{
    //Create CustomOpenApiClient
    string url = $"{Configuration.DOMAIL_URL}/dir/domail-input-rest/";
    var client = new CustomOpenApiClient(url);

    //Create EmailSendAdvancedRequest
    var emailRequest = new EmailSendAdvancedRequest();

    //set 1 recipients
    emailRequest.To = new List<EmailAddressType>() {
            new EmailAddressType() { AddressName = "NameOfEmail", EmailAddress = "
     email@email.com"},
    };
```

```
    emailRequest.ExtId = "100";
    emailRequest.SysId = "200";

    //send reqest throug REST API EmailSendSimpleAsync
    Task<CommunicationId> taskEmailSendSimpleAsync = client.EmailSendAdvancedAsync(
     emailRequest);

    //wait for resposne
    var result = taskEmailSendSimpleAsync.GetAwaiter().GetResult();

    if (result.Id > 0)
        Console.WriteLine($"call POST OK - result.Id={result.Id}");

}
catch (ApiException<RestErrorResponse> e1)
{
    Console.WriteLine($"ApiException: {e1}\nStatusCode: {e1.StatusCode}\nResult.
     ResponseCode: {e1.Result?.ResponseCode}\nResult.Message: {e1.Result?.Message}");
}
catch (ApiException e2)
{
    Console.WriteLine($"ApiException: {e2}, StatusCode: {e2.StatusCode}");
}
catch (Exception ex)
{
    Console.WriteLine($"Exception: {ex}");
}
```

Listing 29: Example for POST request for sending email (advanced) only with required fields in C#

5.3.1.3 Java

TODO

5.3.1.4 Python

TODO

5.3.2 through SOAP

In this part it is a SOAP call

5.3.2.1 XML Request

In the next example 30, there is SOAP POST request only with required fields: We can test this e.g. via Postman.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:p1="
     urn:dominanz.sk/domail">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <p1:emailSendAdvRequest>
        <emailSendRequest>
            <to>
                <name>lukas</name>
                <emailAddress>lukas.vanek+soap@dominanz.sk</emailAddress>
```

```
            </to>
            <scenario>Lukas-SOAP/REST</scenario>
        </emailSendRequest>
        <sysId>34242</sysId>
        <extId>0002</extId>
    </p1:emailSendAdvRequest>
  </soap:Body>
</soap:Envelope>
```

Listing 30: Example for SOAP POST request for sending email (advanced) only with required fields

5.3.2.2 C#

The following listing 31 contains the C# source code that executes a POST request through the SOAP endpoint emailSendAdv.

```
try
{
    //Create Client
    string url = $"{Configuration.DOMAIL_URL}/dws/domail-input-ws/";
    var client = new dominanz_domail_Communication();
    client.Url = url;

    //Create EmailSendAdvRequestType
    var emailRequest = new DoMailGeneratedLibrary.SOAP.EmailSendAdvRequestType();

    //set 1 recipients
    var listArray = new List<DoMailGeneratedLibrary.SOAP.EmailAddressType>() {
            new DoMailGeneratedLibrary.SOAP.EmailAddressType() {
                name = "NameOfEmail",
                emailAddress = "lukas.vanek+soap@dominanz.sk"}
    };

    emailRequest.emailSendRequest = new EmailSendRequestType();
    emailRequest.emailSendRequest.to = listArray.ToArray();
    emailRequest.emailSendRequest.scenario = "Lukas-SOAP/REST";
    emailRequest.sysId = "34242";
    emailRequest.extId = "0002";

    //send reqeust through SOAP emailSendAdv
    var result = client.emailSendAdv(emailRequest);

    if (result.id > 0)
        Console.WriteLine($"call POST OK - result.Id={result.id}");
}
catch (Exception ex)
{
    Console.WriteLine($"Exception: {ex}");
}
```

Listing 31: Example for POST request for sending email (advanced) only with required fields in C#

5.3.2.3 Java

TODO

5.3.2.4 Python

TODO

5.4 Example 4 - Send email - EmailSendAdvanced

This example describes how to call EmailSendAdvanced with all fields.

5.4.1 through REST API

In this part it is a REST API call.

5.4.1.1 JSON

In the next example (32), there is POST request with all fields:

```
{
  "to": [
    {
      "addressName": "NameOfEmail",
      "emailAddress": "email@email.com"
    },
    {
      "addressName": "NameOfEmail2",
      "emailAddress": "email2@email.com"
    }
  ],
  "subject": "Subject of email",
  "htmlBody": "<b>Html body</b>",
  "scenario": "ScenarioName",
  "attachments": [
    {
      "filename": "testFile1",
      "dataHandler": "VG90byBqZSB0ZXN0b3ZhY2lhIHByaWxvaGE=",
      "contentID": "54321",
      "mimeType": "application/text",
      "encoding": "utf-8"
    }
  ],
  "statistics": {
    "group": "testGroupName",
    "operation": "TI212",
    "category": "categoryName",
    "tags": ["tag1", "TEST", "tag2"]
  },
  "testmode": false,
  "extId": "100",
  "sysId": "200",
  "advancedRequestData": {
    "priority": false,
    "duplicity": false
  }
}
```

Listing 32: Example for POST request for sending email (simple) with all fields

5.4.1.2 C#

Complete source file is on the next listening 33 or the whole solution with examples can be downloaded from: TODO: WWW

```
try
{
    string url = $"{Configuration.DOMAIL_URL}/dir/domail-input-rest/";
    var client = new CustomOpenApiClient(url);//Create CustomOpenApiClient
```

```
    //Create EmailSendAdvancedRequest
    var emailRequest = new EmailSendAdvancedRequest();
    //set 2 recipients
    emailRequest.To = new List<EmailAddressType>() {
            new EmailAddressType() { AddressName = "NameOfEmail", EmailAddress = "email@email.com"},
            new EmailAddressType() { AddressName = "NameOfEmail2", EmailAddress = "email2@email.com"},
    };

    var utf8 = new UTF8Encoding();
    emailRequest.Attachments = new List<FileHandlerType>(){
        new FileHandlerType()
        {
            Filename = "testFile1",
            DataHandler = utf8.GetBytes("VG90byBqZSB0ZXN0b3ZhY2lhIHByaWxvaGE="),
            ContentID = "54321",
            MimeType = "application/text",
            Encoding = "UTF-8"
        }
    };
    emailRequest.Statistics = new StatisticsType()
    {
        Group = "testGroupName",
        Operation = "TI212",
        Category = "categoryName",
        Tags = new List<string>() { "tag1", "TEST", "tag2" }
    };
    emailRequest.Subject = "Subject of email";
    emailRequest.HtmlBody = "<b>Html body/b>";
    emailRequest.Scenario = "ScenarioName";
    emailRequest.Testmode = false;
    emailRequest.ExtId = "100";
    emailRequest.SysId = "200";
    emailRequest.AdvancedRequestData = new AdvancedRequestType()
    {
        Priority = false,
        Duplicity = false
    };
    //send reqest throug REST API EmailSendSimpleAsync
    Task<CommunicationId> taskEmailSendSimpleAsync = client.EmailSendAdvancedAsync(emailRequest);

    //wait for resposne
    var result = taskEmailSendSimpleAsync.GetAwaiter().GetResult();

    if (result.Id > 0)
        Console.WriteLine($"call POST OK - result.Id={result.Id}");
}
catch (ApiException<RestErrorResponse> e1)
{
    Console.WriteLine($"ApiException: {e1}\nStatusCode: {e1.StatusCode}\nResult.ResponseCode: {e1.Result?.ResponseCode}\nResult.Message: {e1.Result?.Message}");
}
catch (ApiException e2)
{
    Console.WriteLine($"ApiException: {e2}, StatusCode: {e2.StatusCode}");
}
catch (Exception ex)
{
    Console.WriteLine($"Exception: {ex}");
```

```
}
```

Listing 33: Example for POST request for sending email (advanced) with all fields in C#

5.4.1.3 Java

TODO

5.4.1.4 Python

TODO

5.4.2 through SOAP

In this part it is a SOAP call.

5.4.2.1 XML Request

In the next example 34, there is SOAP POST request only with all fields: We can test this e.g. via Postman.

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:p1="
    urn:dominanz.sk/domail">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <p1:emailSendAdvRequest>
        <emailSendRequest>
            <to>
                <name>NameOfEmail1</name>
                <emailAddress>lukas.vanek+soap1@dominanz.sk</emailAddress>
            </to>
            <to>
                <name>NameOfEmail2</name>
                <emailAddress>lukas.vanek+soap2@dominanz.sk</emailAddress>
            </to>
            <scenario>Lukas-SOAP/REST</scenario>
            <testMode>false</testMode>
            <statistics>
                <group>testGroupName</group>
                <operation>TI212</operation>
                <category>categoryName</category>
                <tag>tag1</tag>
                <tag>TEST</tag>
                <tag>tag2</tag>
            </statistics>
        </emailSendRequest>
        <sysId>34242</sysId>
        <extId>0002</extId>
        <advreqdata>
            <priority>true</priority>
            <duplicity>true</duplicity>
            <!-- <sendtime></sendtime> -->
        </advreqdata>
    </p1:emailSendAdvRequest>
  </soap:Body>
</soap:Envelope>
```

Listing 34: Example for SOAP POST request for sending email (advanced) only with all fields

5.4.2.2 C#

The following listing 35 contains the C# source code that executes a POST request through the SOAP endpoint emailSendAdv.

```
try
{
    //Create Client
    string url = $"{Configuration.DOMAIL_URL}/dws/domail-input-ws/";
    var client = new dominanz_domail_Communication();
    client.Url = url;

    //Create EmailSendAdvRequestType
    var emailRequest = new DoMailGeneratedLibrary.SOAP.EmailSendAdvRequestType();

    //set 2 recipients
    var listArray = new List<DoMailGeneratedLibrary.SOAP.EmailAddressType>() {
            new DoMailGeneratedLibrary.SOAP.EmailAddressType() {
                name = "NameOfEmail",
                emailAddress = "lukas.vanek+soap1@dominanz.sk"},
            new DoMailGeneratedLibrary.SOAP.EmailAddressType() {
                name = "NameOfEmail2",
                emailAddress = "lukas.vanek+soap2@dominanz.sk"}
    };

    emailRequest.emailSendRequest = new EmailSendRequestType();
    emailRequest.emailSendRequest.to = listArray.ToArray();
    emailRequest.emailSendRequest.scenario = "Lukas-SOAP/REST";
    emailRequest.emailSendRequest.subject = "Subject of email";
    emailRequest.emailSendRequest.htmlBody = "<b>Html body/b>";
    emailRequest.emailSendRequest.testMode = false;
    emailRequest.sysId = "34242";
    emailRequest.extId = "0003";

    emailRequest.emailSendRequest.statistics = new DoMailGeneratedLibrary.SOAP.
     StatisticsType()
    {
        group = "testGroupName",
        operation = "TI212",
        category = "categoryName",
        tag = new String[] { "tag1", "TEST" , "tag2"}
    };

    emailRequest.advreqdata = new AdvReqType()
    {
        priority = true,
        duplicity = true
    };

    //send reqeust through SOAP emailSendAdv
    var result = client.emailSendAdv(emailRequest);

    if (result.id > 0)
        Console.WriteLine($"call POST OK - result.Id={result.id}");
}
catch (Exception ex)
```

```
{
    Console.WriteLine($"Exception: {ex}");
}
```

Listing 35: Example for POST request for sending email (advanced) only with all fields in C#

5.4.2.3 Java

TODO

5.4.2.4 Python

TODO

5.5 Example 5 - Get detail of communication

This example describes how to call CommunicationGet.

5.5.1 through REST API

In this part it is a REST API call.

5.5.1.1 JSON

In this example, no BODY is sent to the POST request to call endpoint:
http:\\{yourUrl}/dir/domail-input-rest/communication/1221981/detail

After a successful call it returns a response 36:

```
{
    "id": 1221981,
    "rcvTime": "2023-08-16T06:42:30.824",
    "sysId": "34242",
    "extId": "0003",
    "lastProcessingState": "DISPATCHED",
    "lastDeliveryState": "DELIVERY_UNKNOWN",
    "scenario": "Lukas-SOAP/REST",
    "testmode": true,
    "statistics": {
        "group": "testGroupName",
        "operation": "TI212",
        "category": "categoryName",
        "tags": [
            "tag1",
            "TEST",
            "tag2"
        ]
    },
    "advancedRequestData": {
        "priority": true,
        "duplicity": false,
        "sendTime": "2023-08-16T06:42:30.824"
    },
    "runs": [
        {
            "runId": 1,
            "procTime": "2023-08-16T06:42:30.869",
            "scenarioName": null,
            "processingState": "DISPATCHED",
            "deliveryState": "DELIVERY_UNKNOWN",
            "sentEmails": [
                {
                    "addressNumber": 1,
```

```
                    "addressName": "NameOfEmail",
                    "emailAddress": "lukas.vanek+soap1@dominanz.sk",
                    "processingState": "DISPATCHED",
                    "deliveryState": "DELIVERY_UNKNOWN",
                    "messageId": "domic.1221981.1.1.938097447.4.1692168151871@domail-test
    ↪                 ",
                    "sendTime": "2023-08-16T06:42:32.56"
                },
                {
                    "addressNumber": 2,
                    "addressName": "NameOfEmail2",
                    "emailAddress": "lukas.vanek+soap2@dominanz.sk",
                    "processingState": "DISPATCHED",
                    "deliveryState": "DELIVERY_UNKNOWN",
                    "messageId": "domic.1221981.2.1.687338147.5.1692168152692@domail-test
    ↪                 ",
                    "sendTime": "2023-08-16T06:42:32.912"
                }
            ]
        }
    ]
}
```

Listing 36: Example for POST response for getting detail of communication

5.5.1.2 C#

Complete source file is on the next listening 37 or the whole solution with examples can be downloaded from: TODO: WWW

```
...

//send reqest throug REST API DetailAsync
Task<CommunicationDetail> taskCommunicationDetail = client.DetailAsync(1221981);

//wait for resposne
var result = taskCommunicationDetail.GetAwaiter().GetResult();

if (result.Id > 0)
    Console.WriteLine($"call POST OK - result.Id={result.Id}");

...
```

Listing 37: Part of example for POST request for get deail of communication in C#

5.5.1.3 Java

TODO

5.5.1.4 Python

TODO

5.5.2 through SOAP

In this part it is a SOAP call.

5.5.2.1 XML Request

In the next example 38, there is SOAP POST request: We can test this e.g. via Postman.

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:p1="
    urn:dominanz.sk/domail">
  <soap:Header></soap:Header>
  <soap:Body>
    <p1:communicationGetRequest>
        <id>1221981</id>
    </p1:communicationGetRequest>
  </soap:Body>
</soap:Envelope>
```

Listing 38: Example for SOAP POST request for get detail of communication

After a successful call it returns a SOAP response: 39

```xml
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
      <ns3:communicationGetResponse xmlns:ns3="urn:dominanz.sk/domail">
          <detail>
              <id>1221981</id>
              <sysId>34242</sysId>
              <extId>0003</extId>
              <lastState>
                  <processingState>DISPATCHED</processingState>
                  <deliveryState>DELIVERY_UNKNOWN</deliveryState>
              </lastState>
              <scenario>Lukas-SOAP/REST</scenario>
              <statistics>
                  <group>testGroupName</group>
                  <operation>TI212</operation>
                  <category>categoryName</category>
                  <tag>tag1</tag>
                  <tag>TEST</tag>
                  <tag>tag2</tag>
              </statistics>
              <testMode>true</testMode>
              <rcvTime>2023-08-16T06:42:30.824Z</rcvTime>
              <advancedReqData>
                  <priority>true</priority>
                  <duplicity>false</duplicity>
                  <sendTime>2023-08-16T06:42:30.824Z</sendTime>
              </advancedReqData>
              <runs>
                  <runId>1</runId>
                  <procTime>2023-08-16T06:42:30.869Z</procTime>
                  <state>
                      <processingState>DISPATCHED</processingState>
                      <deliveryState>DELIVERY_UNKNOWN</deliveryState>
                  </state>
                  <sentEmails>
                      <addressNumber>1</addressNumber>
                      <address>
                          <name>NameOfEmail</name>
                          <emailAddress>lukas.vanek+soap1@dominanz.sk</emailAddress>
                      </address>
                      <state>
                          <processingState>DISPATCHED</processingState>
                          <deliveryState>DELIVERY_UNKNOWN</deliveryState>
                      </state>
                      <messageId>domic.1221981.1.1.938097447.4.1692168151871@domail-test
                              </messageId>
```

```
                    <sendTime>2023-08-16T06:42:32.560Z</sendTime>
                </sentEmails>
                <sentEmails>
                    <addressNumber>2</addressNumber>
                    <address>
                        <name>NameOfEmail2</name>
                        <emailAddress>lukas.vanek+soap2@dominanz.sk</emailAddress>
                    </address>
                    <state>
                        <processingState>DISPATCHED</processingState>
                        <deliveryState>DELIVERY_UNKNOWN</deliveryState>
                    </state>
                    <messageId>domic.1221981.2.1.687338147.5.1692168152692@domail-test
                            </messageId>
                    <sendTime>2023-08-16T06:42:32.912Z</sendTime>
                </sentEmails>
            </runs>
        </detail>
    </ns3:communicationGetResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 39: Example for SOAP POST response for get detail of communication

5.5.2.2 C#

The following listing 40 contains part of the C# source code that executes a POST request through the SOAP endpoint communicationGet.

```
...
CommunicationGetRequestType communicationGetRequest = new CommunicationGetRequestType()
{
    id = 1221981
};

CommunicationDetailType[] result = client.communicationGet(communicationGetRequest);
...
```

Listing 40: Example for POST request for get detail of communication in C#

5.5.2.3 Java

TODO

5.5.2.4 Python

TODO

5.6 Example 6 - Get state of communication

This example describes how to call CommunicationGetState with all fields.

5.6.1 through REST API

In this part it is a REST API call.

5.6.1.1 JSON

In this example, no BODY is sent to the POST request to call endpoint:
http:\\{yourUrl}/dir/domail-input-rest/communication/1221981/state
After a successful call it returns a response 41:

```
{
    "lastProcessingState": "DISPATCHED",
    "lastDeliveryState": "DELIVERY_UNKNOWN",
    "id": 1221981,
    "runs": [
        {
            "processingState": "DISPATCHED",
            "deliveryState": "DELIVERY_UNKNOWN",
            "runId": 1,
            "sentEmails": [
                {
                    "addressNumber": 1,
                    "addressName": "NameOfEmail",
                    "emailAddress": "lukas.vanek+soap1@dominanz.sk",
                    "processingState": "DISPATCHED",
                    "deliveryState": "DELIVERY_UNKNOWN"
                },
                {
                    "addressNumber": 2,
                    "addressName": "NameOfEmail2",
                    "emailAddress": "lukas.vanek+soap2@dominanz.sk",
                    "processingState": "DISPATCHED",
                    "deliveryState": "DELIVERY_UNKNOWN"
                }
            ]
        }
    ]
}
```

Listing 41: Example for POST response for getting state of communication

5.6.1.2 C#

Complete source file is on the next listening 42 or the whole solution with examples can be downloaded from: TODO: WWW

```
...
//send reqest throug REST API StateAsync
Task<CommunicationStateType> taskCommunicationState = client.StateAsync(1221981);

//wait for resposne
var result = taskCommunicationState.GetAwaiter().GetResult();
...
```

Listing 42: Example for SOAP POST request for get state of communication in C#

5.6.1.3 Java

TODO

5.6.1.4 Python

TODO

5.6.2 through SOAP

In this part it is a SOAP call.

5.6.2.1 XML Request

In the next example 43, there is SOAP POST request: We can test this e.g. via Postman.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:p1="
     urn:dominanz.sk/domail">
    <soap:Header></soap:Header>
    <soap:Body>
    <p1:communicationGetStateRequest>
        <id>1221981</id>
    </p1:communicationGetStateRequest>
    </soap:Body>
</soap:Envelope>
```

Listing 43: Example for SOAP POST request for getting state of communication

After a successful call it returns a SOAP response: 44

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns3:communicationGetStateResponse xmlns:ns3="urn:dominanz.sk/domail">
            <state>
                <id>1221981</id>
                <lastState>
                    <processingState>DISPATCHED</processingState>
                    <deliveryState>DELIVERY_UNKNOWN</deliveryState>
                </lastState>
                <runs>
                    <runId>1</runId>
                    <state>
                        <processingState>DISPATCHED</processingState>
                        <deliveryState>DELIVERY_UNKNOWN</deliveryState>
                    </state>
                    <sentEmails>
                        <addressNumber>1</addressNumber>
                        <address>
                            <name>NameOfEmail</name>
                            <emailAddress>lukas.vanek+soap1@dominanz.sk</emailAddress>
                        </address>
                        <state>
                            <processingState>DISPATCHED</processingState>
                            <deliveryState>DELIVERY_UNKNOWN</deliveryState>
                        </state>
                    </sentEmails>
                    <sentEmails>
                        <addressNumber>2</addressNumber>
                        <address>
                            <name>NameOfEmail2</name>
                            <emailAddress>lukas.vanek+soap2@dominanz.sk</emailAddress>
                        </address>
                        <state>
                            <processingState>DISPATCHED</processingState>
                            <deliveryState>DELIVERY_UNKNOWN</deliveryState>
                        </state>
                    </sentEmails>
                </runs>
            </state>
        </ns3:communicationGetStateResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 44: Example for SOAP POST response for getting state of communication

5.6.2.2 C#

The following listing 45 contains the part of C# source code that executes a POST request through the SOAP endpoint communicationGetState.

```
...
var communicationGetStateRequest = new DoMailGeneratedLibrary.SOAP.
    CommunicationGetStateRequestType()
{
    id = 1221981
};

DoMailGeneratedLibrary.SOAP.CommunicationStateType[] result = client.
    communicationGetState(communicationGetStateRequest);
...
```

Listing 45: Example for POST request for getting state of communication in C#

5.6.2.3 Java

TODO

5.6.2.4 Python

TODO

5.7 Reference guide

Name **Description**	**Script**
Attachment is Compare whether attachment (name, type) is same or contains (using * and ? characters) specified text	attachment == "*.png"
Content of email is Compare whether content of email is same or contains (using * and ? characters) specified text	content == "*@domail.eu"
Content request is Compare whether content of request is same or contains (using * and ? characters) specified text	content == "*@domail.eu"
Count attachments Compare count of attachments	attachment.count >= 0
Count "BCC" recipients Compare count of recipients "BCC"	bcc.count > 0
Count "CC" recipients Compare count of recipients "CC"	cc.count > 0
Count of attachments Compare count of attachments	attachment.count >= 0

Name **Description**	**Script**
Count "TO" recipients Compare count of recipients "TO"	to.count > 0
Domain "TO" is Compare whether domain of recipient "TO" is same or contains (using * and ? characters) specified text	to.domain == "domail.eu"
Field "category" is Compare whether field "category" is same or contains (using * and ? characters) specified text	category == "domail.eu"
Field "duplicity" is Compare whether field "duplicity" has same logical value	duplicity == true
Field "extId" is Compare whether field "extId" is same or contains (using * and ? characters) specified text	extId == "domail.eu"
Field "group" is Compare whether field "group" is same or contains (using * and ? characters) specified text	group == "domail.eu"
Field "instance" is Compare whether field "instance" is same or contains (using * and ? characters) specified text	instance == "domail.eu"
Field "operation" is Compare whether field "operation" is same or contains (using * and ? characters) specified text	operation == "domail.eu"
Field "priority" is Compare whether field "priority" has same logical value	priority == true
Field "sysId" is Compare whether field "sysId" is same or contains (using * and ? characters) specified text	sysId == "domail.eu"
Field "tags" is Compare whether field "tags" is same or contains (using * and ? characters) specified text	tags == "domail.eu"
Field "testMode" is Compare whether field "testMode" has same logical value	testMode == true
Filter attachments Filter attachments by requested attributes and evaluate condition	attachment.filename("*.png"). content-Type("image/*").count >= 1
Filter attachments Filter attachments by requested attributes and evaluates condition	attachment.filename("*.png"). content-Type("image/*").count >= 1
Logical operator "and" Condition is true only if left and right side of condition are true	and
Logical operator "and" Condition is true only if left and right side of condition is true	and
Logical operator "negation" Negate condition in braces - true <-> false	!()
Logical operator "or" Condition is true if left or right side of condition is true	or

Name Description	Script
Logical operator "or" Condition true is if left or right side of condition is true	or
Logical operator "or" Condition true is if true is left or right side of condition	or
Recipient "BCC" is Compare whether recipient "BCC" is same or contains (using * and ? characters) specified text	bcc == "*@domail.eu"
Recipient "CC" is Compare whether recipient "CC" is same or contains (using * and ? characters) specified text	cc == "*@domail.eu"
Recipient "TO" is Compare whether recipient "TO" is same or contains (using * and ? characters) specified text	to == "*@domail.eu"
Sender is Compare whether sender is same or contains (using * and ? characters) specified text	from == "*@domail.eu"
Size of attachments Compare size of attachments	attachment.size >= 1MB
Size of email is Compare total size of email	size >= 1MB
Size of request is Compare total size of request	size >= 1MB
Subject is Compare whether subject is same or contains (using * and ? characters) specified text	subject == "*@domail.eu"
Type of attachments is Compare whether type of attachment (mime type) is same or contains (using * and ? characters) specified text	attachment.contentType == "image/*"

5.8 Scenarios selection scripts (conditions)

This part is relevant only for the communication channel SMTP. The scripts use or implement Śpring Expression language:
The Spring Expression Language (SpEL for short) is a powerful expression language that supports querying and manipulating an object graph at runtime. The language syntax is similar to Unified EL but offers additional features, most notably method invocation and basic string templating functionality.
link to website: Spring Expression Language (SpEL)

5.8.1 Compose conditions for scenario selection

TODO

5.8.1.1 Basic conditions

TODO

5.8.1.2 Advanced conditions

TODO

5.9 Processing scripts

5.9.1 ECMA/Javascript

ECMAScript [**ecmaScript**] is a standard for scripting languages, including JavaScript, JScript, and ActionScript. It is also best known as a JavaScript standard intended to ensure the interoperability of web pages across different web browsers.It is standardized by Ecma International in the document ECMA-262.

ECMAScript is commonly used for clientside scripting on the World Wide Web, and it is increasingly being used to write server-side applications and services using Node.js and other runtime environments.
In the following ECMA/Javascript code block 46, there is an example of a default script, which you can of course set yourself according to your requirements.

```
to();           // set address "TO" from request
text();         // set text email from request
html();         // set html text emailu from request
subject();      // set subject from request
from();         // set address "FROM" from request
returnPath();   // set returnPath from request
replyTo();      // set replyTo from request

smtpServer();   // set SMTP server
attachment();   // set attachments from request
trackPixel();   // set track pixel, if email is in HTML mode
```

Listing 46: Example of default ECMA/Javascript for processing email

5.9.2 Reference guide for processing scripts

5.9.2.1 Work with attachments

Script / Description
attachment.add("filename"); Add attachment from file system. Directory with attachments is specified by parameter of scenario
attachment.add("filename","name"); Add attachment from file system. Name of attachment in email may be different. Directory with attachments is specified by parameter of scenario.
attachment.addFromGallery("filename"); Add attachment from gallery.
attachment.addFromGallery("filename","name"); Add attachment from gallery. Name of attachment in email may be different.
attachment.add(index); Add attachment from request on specified index

5.9.2.2 Work with 'Bcc' addresses

Script / Description
bcc(); set 'bcc' adress(es) from settings of scenario
bcc.add("address"); add single address of type Bcc
bcc.add("address1", "address2", ...); add multiple addresses of type Bcc
bcc.addEmailWithName("email", "name"); add single address with name of type Bcc
bcc.clear(); remove all addresses of type Bcc
bcc.count(); return count of addresses set of type Bcc
bcc.set("address"); set single address of type Bcc
bcc.set("address1", "address2", ...); set multiple addresses of type Bcc
bcc.setEmailWithName("email", "name"); set single address with name of type Bcc

5.9.2.3 Work with 'Cc' addresses

Script / Description
cc(); set 'cc' adress(es) from settings of scenario
cc.add("address"); add single address of type Cc
cc.add("address1", "address2", ...); add multiple addresses of type Cc
cc.addEmailWithName("email", "name"); add single address with name of type Cc
cc.clear(); remove all addresses of type Cc
cc.count(); return count of addresses set of type Cc
cc.set("address"); set single address of type Cc
cc.set("address1", "address2", ...); set multiple addresses of type Cc
cc.setEmailWithName("email", "name"); set single address with name of type Cc

5.9.2.4 Context - variables for processing message

Script / Description
ctx.attachments; get List<Attachment<?» object - list of attachments that will be added to email after processing ends
ctx.fail("Error message"); Mark processing as invalid with given description
ctx.getAttachmentsByExtension(".pdf"); get List<Attachment<?» object - list of attachments that will be added to email after processing ends that have specified extension in their name
ctx.getDbEmail(); get srv_emails object - database entry of single email
ctx.getDbReq(); get srv_emlreq object - database entry of request
ctx.getDbReqRun(); get srv_procrun object - database entry of run
ctx.id; get IdNumRunid object - id of actual processing composed of id of request, num - index of email and runid - id of run
ctx.isFailed(); returns true if error ocurred during processing, otherwise return false
ctx.message; get MailMessage object for work with generated message
ctx.message.getMessage(); get SMTPMessage object - generated email
ctx.params; get Map<String, Object> object - map of parameters set during processing
ctx.req; get CreateCommunicationRequestI object - request to send email
ctx.scenarioParams; get Map<String, String> object - map of parameters set for actual scenario
ctx.scriptConstants; get Map<String, String> object - map of set constants
ctx.service; get services to work with
ctx.service.db; get DbService object - service for work with database
ctx.service.fs; get FsService object - service for work with file system

5.9.2.5 Sign email with DKIM

Script / Description
dkim(); if parameter of scenario is set, sign email with DKIM (while sending)
dkim.use(); Sign email with DKIM. Parameter of scenario specifies alias of certificate

5.9.2.6 Work with 'From' addresses

Script / Description
from(); set 'from' address from settings of scenario
from.clear(); remove all addresses of type From
from.count(); return count of addresses set of type From
from.set("address"); set single address of type From
from.setEmailWithName("email", "name"); set single address with name of type From

5.9.2.7 Work with text/html content of message

Script / Description
html(); use HTML content from request
html.set("html"); set specified HTML content of email

5.9.2.8 Logging to database and similar

Script / Description
log.logError("message"); log ERROR message
log.logInfo("message"); log INFO message
log.log("level","message"); log message with given level

5.9.2.9 Set priority

Script / Description
priority(); Set message as priority/nonpriority, based on settings of scenario. If scenario does not have settings keep original value
priority.set(); Set message as priority - send it to priority queue
priority.set(true/false); Set message as priority/nonpriority

5.9.2.10 Work with 'qr' code

Script / Description
qr(); Set qr code - image in html part of email. QR code is read from iContent part of request. If does not contain html or html does not contain qr section then do nothing
qr.addToEndHtmlPart(); Set qr code - image in html part of email. QR code is read from iContent part of request. If does not contain html then do nothing. If html does not contain qr section then it is added at the end
qr.addToEndHtmlPart("QR code value"); Set given qr code - image in html part of email. If does not contain html then do nothing. If html does not contain qr section then it is added at the end
qr.generate(); Set qr code - image in html part of email. QR code is read from iContent part of request. Email must contain html and html must contain qr section, otherwise it is error
qr.generate("QR code value"); Set given qr code - image in html part of email. Email must contain html and html must contain qr section, otherwise it is error
qr.update("QR code value"); Set given qr code - image in html part of email. If does not contain html or html does not contain qr section then do nothing

5.9.2.11 Work with 'Reply to' addresses

Script / Description
replyTo.add("address"); add single address of type ReplyTo
replyTo.add("address1", "address2", ...); add multiple addresses of type ReplyTo
replyTo.addEmailWithName("email", "name"); add single address with name of type ReplyTo
replyTo.clear(); remove all addresses of type ReplyTo
replyTo.count(); return count of addresses set of type ReplyTo
replyTo.set("address"); set single address of type ReplyTo
replyTo.set("address1", "address2", ...); set multiple addresses of type ReplyTo
replyTo.setEmailWithName("email", "name"); set single address with name of type ReplyTo

5.9.2.12 Monitor undelivered message

Script / Description
reportUndelivered.sendEmail(); Prepare email with report of undelivered emails to sending.

5.9.2.13 Work with 'returnPath'

Script / Description
returnPath(); set 'return-path' address from settings of scenario
returnPath.set("address"); set given 'return-path' address

5.9.2.14 Sign email by certificate

Script / Description
signEmail(); Sign email. Sign by parameters of scenario. Attachments must already be added by attachment(Work with attachments) plugin!

5.9.2.15 Sign PDF attachments

Script / Description
signPdf(); Sign all PDF attachments. Sign by parameters of scenario. Attachments must be already added by attachments plugin!

5.9.2.16 Set SMTP server for sending email

Script / Description
smtpServer(); use SMTP server from settings of scenario
smtpServer.use("smtpServerId"); set id of SMTP server for sending email

5.9.2.17 Perform standard processing based on request and scenario settings

Script / Description
standard(); perform standard processing on plugins, f.e. to(); subject();
standard.except(module1, module2...); perform standard processing on all plugins except named plugins
standard.standard(module1, module2...); perform standard processing on named plugins

5.9.2.18 Set subject

Script / Description
subject(); Set subject of email from request
subject.set("Subject of email"); Set subject email

5.9.2.19 Work with text/plain content of message

Script / Description
text(); set text content of email from request
text.set("Text of email"); set text content of email

5.9.2.20 Work with 'To' addresses

Script / Description
to(); set 'to' adress(es) from request and from settings of scenario or from request
to.add("address"); add single address of type To
to.add("address1", "address2", ...); add multiple addresses of type To
to.addEmailWithName("email", "name"); add single address with name of type To
to.clear(); remove all addresses of type To
to.count(); return count of addresses set of type To
to.set("address"); set single address of type To

to.set("address1", "address2", ...); set multiple addresses of type To
to.setEmailWithName("email", "name"); set single address with name of type To

5.9.2.21 Work with 'track pixel'

Script / Description
trackPixel.add(); Set track-pixel image in html part of email. If does not contain html then do nothing. If html does not contain track pixel section then it is added at the end
trackPixel.update(); Set track-pixel image in html part of email. If does not contain html or html does not contain track pixel section then do nothing

5.9.2.22 Validation of message (inputs, outputs)

Script / Description
validate.message(); Validate content of email

5.9.3 To, CC, BCC, Subject

In the following ECMA/Javascript code block 47, we can modify the script that can be used in some scenario. The goal of the script is to modify or add some addresses to the "TO", "CC" or "BCC" fields.

```
to();           // set address "TO" from request
text();         // set text email from request
html();         // set html text emailu from request
subject();      // set subject from request
from();         // set address "FROM" from request
returnPath();   // set returnPath from request
replyTo();      // set replyTo from request

smtpServer();   // set SMTP server
attachment();   // set attachments from request
trackPixel();   // set track pixel, if email is in HTML mode

to.add("example-to@dominanz.sk"); // add fixed address to "TO"

cc();           // set address "CC" from request
cc.add("example-cc@dominanz.sk", "example-cc2@dominanz.sk"); // add fixed addresses to "
    CC"

bcc();          // set address "BCC" from request
bcc.add("example-bcc@dominanz.sk");   // add fixed address to "BCC"
```

Listing 47: ECMA/Javascript for changing fields TO, CC, BCC

This example can be used if we want every communication that will be sent using this script to have a fixed address added, to which we also want to send an email.

On the following figure 40, you can see after processing the email communication how the TO, CC, BCC defined in the script have been added to the EML source.

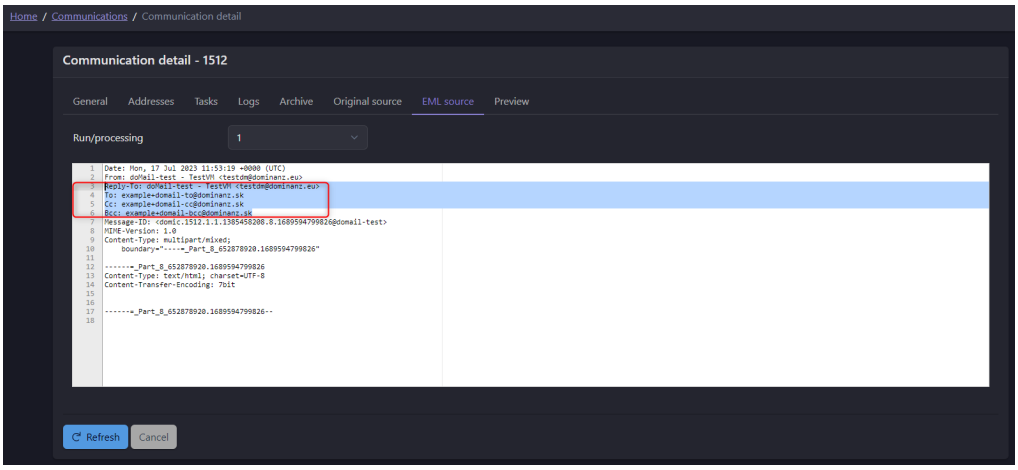

Figure 40: Communication detail - EML source - ExampleEcmaScriptToCcBcc

Another possible way (listing 48) of use can be , if we use the server for example in DEV or UAT mode, so we can overwrite all addresses and set our desired address.

```
...
to.clear();     // clear address "TO" from request
cc.clear();     // clear address "CC" from request
bcc.clear();    // clear address "BCC" from request
to.set("example-to-TEST@dominanz.sk");  // SET fixed address to "TO"
...
```

Listing 48: ECMA/Javascript for changing field TO in test mode

5.9.4 PlainTextBody, HtmlTextBody

In the following example (listing 49) we will show how we can add e.g. a signature to each sending communication in plaing text.

```
...
var plainText = ctx.req.getPlainText();
if(plainText != null && plainText != ""){
    plainText = plainText + '\n\r Pridany podpis na koniec';
    text.set(plainText);
}
...
```

Listing 49: ECMA/Javascript for adding text to body - PLAIN

Result for this request JSON (listing 50):

```
{
    "to": [
        {
            "addressName": "Lukas Vanek",
            "emailAddress": "lukas.vanek+domail@dominanz.sk"
        }
```

```
    ],
    "plainTextBody": "Povodny text emailu",
    "testmode": false
}
```

Listing 50: Example for POST request for sending email with adding plain text in script

Result EML:

```
Date: Thu, 20 Jul 2023 06:28:27 +0000 (UTC)
From: doMail-test - TestVM <testdm@dominanz.eu>
Reply-To: doMail-test - TestVM <testdm@dominanz.eu>
To: Lukas Vanek <lukas.vanek+domail@dominanz.sk>
Message-ID: <domic.1559.1.1.570591244.57.1689834507143@domail-test>
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="----=_Part_57_608698271.1689834507143"

------=_Part_57_608698271.1689834507143
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 7bit

Povodny text emailu

 Pridany podpis na koniec
------=_Part_57_608698271.1689834507143--
```

Figure 41: Communication detail - EML source - Adding PlainText to Body

In the following example (listing 51) we will show how we can add e.g. a signature to each sending communication in HTML.

```
...
var htmlVal = ctx.req.getHtmlText();
if(htmlVal != null && htmlVal != ""){
  htmlVal = htmlVal + '<p> </p><p><strong
     >----------------------------------------</strong></p><p><strong>FirstName LastName
     , Company</strong></p>'
  html.set(htmlVal);
}
...
```

Listing 51: ECMA/Javascript for adding text to body - HTML

Result for this request JSON (listing 52):

```
{
    "to": [
        {
            "addressName": "Lukas Vanek",
            "emailAddress": "lukas.vanek+domail@dominanz.sk"
        }
    ],
    "htmlBody": "Povodny HTML text emailu\r\n",
    "testmode": false
}
```

Listing 52: Example for POST request for sending email with adding HTML text in script

Result EML:

```
Date: Thu, 20 Jul 2023 06:46:03 +0000 (UTC)
From: doMail-test - TestVM <testdm@dominanz.eu>
Reply-To: doMail-test - TestVM <testdm@dominanz.eu>
To: Lukas Vanek <lukas.vanek+domail@dominanz.sk>
Message-ID: <domic.1561.1.1.86241720.59.1689835563224@domail-test>
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="----=_Part_59_73219858.1689835563224"

------=_Part_59_73219858.1689835563224
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: 7bit

Povodny HTML text emailu
<p> </p><p><strong>----------------------------------------</strong></p><p><strong>FirstName LastName, Company</strong></p>
------=_Part_59_73219858.1689835563224--
```

Figure 42: Communication detail - EML source - Adding HtmlText to Body

5.9.5 Track pixel

In the following example (listing 53) we will show how we can add track pixel to html body.

```
...
html();
trackPixel();
trackPixel.add(); //add track prixel to HTML email
...
```

Listing 53: ECMA/Javascript for adding track pixel

Result for this request JSON (listing 54):

```
{
    "to": [
        {
            "addressName": "Lukas Vanek",
            "emailAddress": "lukas.vanek+domail@dominanz.sk"
        }
    ],
    "htmlBody": "<b>Text emailu</b>",
    "testmode": false
}
```

Listing 54: Example for POST request for sending email with template only with adding track pisel in script

Result EML:

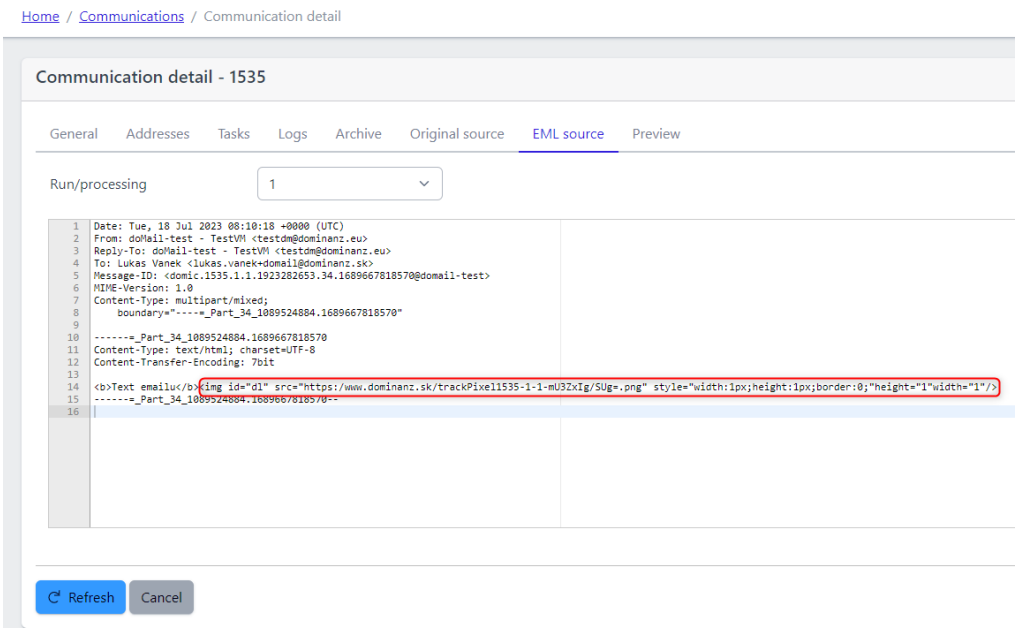

Figure 43: Communication detail - EML source - ExampleEcmaScriptAddingTrackPixel

In case the Track pixel server is running, when the user is reading the email, system Domail will consider the email as delivered and read.

5.9.6 QR codes

5.9.6.1 Add new QR code to end of email

In the following example (listing 55) we will show how we can add generated new QR code to end of HTML email.

```
...
qr.addToEndHtmlPart("www.dominanz.sk");
...
```

Listing 55: ECMA/Javascript for adding new generated QR code

Result for this request JSON (listing 56):

```
{
    "to": [
        {
            "addressName": "Lukas Vanek",
            "emailAddress": "lukas.vanek+domail@dominanz.sk"
        }
    ],
    "htmlBody": "<b>Text emailu</b>\r\n",
    "testmode": false
}
```

Listing 56: Example for POST request for sending email with generated new QR code

Result (44) EML source:

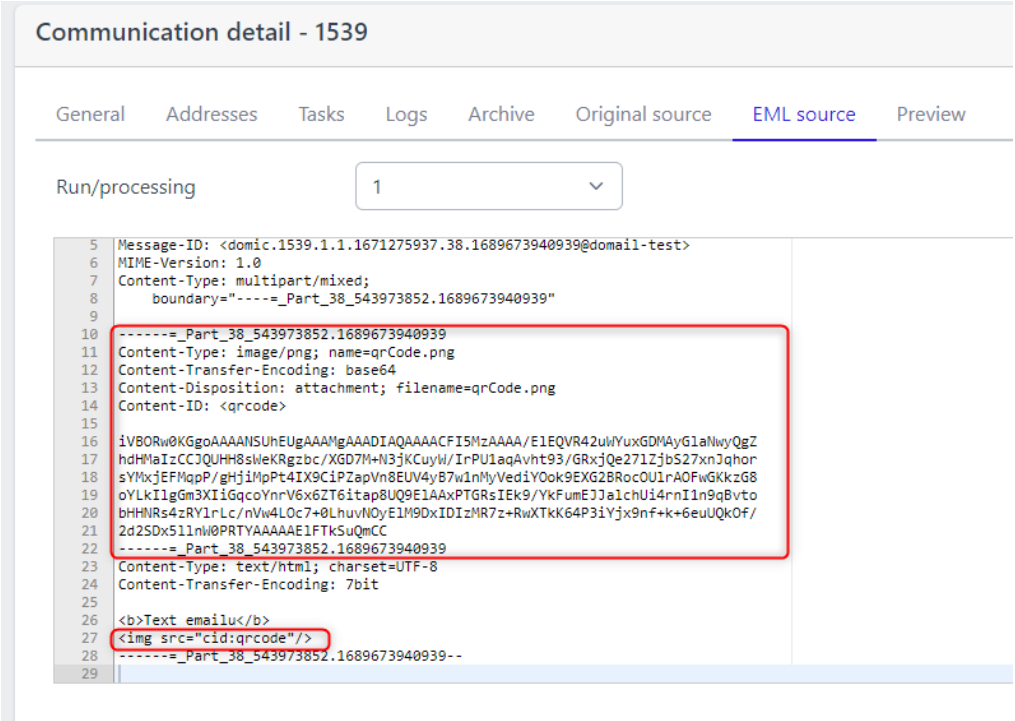

Figure 44: Communication detail - ExampleEcmaScript QR AddToEndHtmlPart EML

Result (45) in Email client Outlook:

Figure 45: Outlook - ExampleEcmaScript QR AddToEndHtmlPart

5.9.6.2 Add QR code from iContent

TODO

5.9.7 Attachments

TODO: spomenut CONFIG parameter kde sa definuje nazov suboru...

5.9.7.1 Add attachments from the request

In the following example (listing 57) we will show how we can add attachment from the request.

```
...
attachment();// attachments from request
...
```

Listing 57: ECMA/Javascript for adding attachments from the request

TODO: priklad s prilozenim prilohy

5.9.7.2 Add attachment from Gallery

In the following example (listing 58) we will show how we can add attachment from galery.

```
...
attachment();// attachments from request
attachment.addFromGallery("read.me","MyCustomReadMe");
...
```

Listing 58: ECMA/Javascript for adding attachment from galery

There is result in following figure 46:

```
Date: Wed, 26 Jul 2023 12:26:35 +0000 (UTC)
From: doMail-test - TestVM <testdm@dominanz.eu>
Reply-To: doMail-test - TestVM <testdm@dominanz.eu>
To: Lukas Vanek <lukas.vanek+domail@dominanz.sk>
Message-ID: <domic.1623.1.1.535931458.2.1690374394932@domail-test>
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="----=_Part_2_782191838.1690374394932"

------=_Part_2_782191838.1690374394932
Content-Type: application/octet-stream; name=MyCustomReadMe
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename=MyCustomReadMe
Content-ID: <MyCustomReadMe>

Example of readme file....
------=_Part_2_782191838.1690374394932--
```

Figure 46: Adding attachment from Domail gallery

5.9.8 Certificates, DKIM

5.9.8.1 Domail certificates

In 'Settings -> Certificates -> Domail certificates' you can manage the certificates that are used for signing attachments and emails. On the following figure 47 contains example of our certificate.

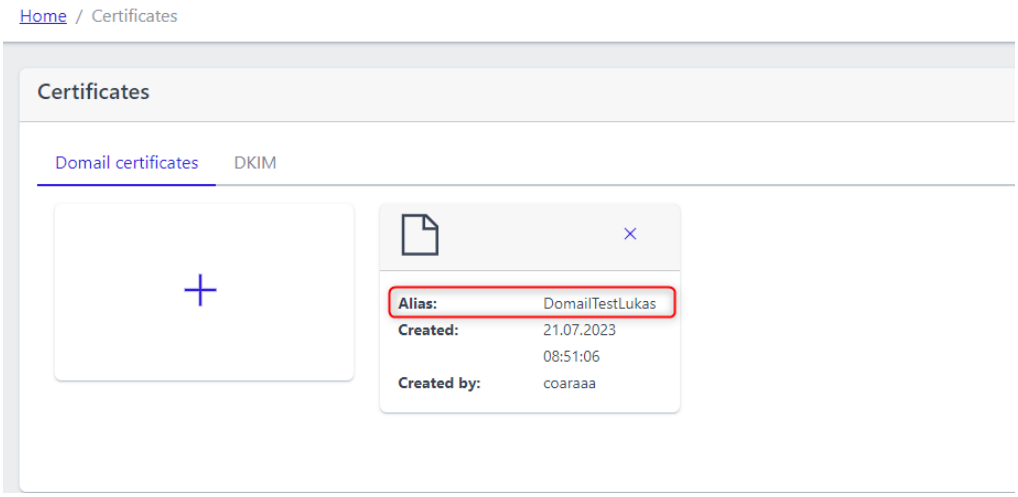

Figure 47: Uploaded certificate for signing attachment

5.9.8.2 DKIM

What DKIM is for? It is a technology for increasing the trustworthiness of emails, which helps to detect spoofed messages. The sent message is signed by the SMTP server with the private key of the sender's domain. This signature is stored in the email header. The receiving server compares this signature with the public key stored in the domain's DNS records. By matching the signature, it is proven that the email actually originated from the sender's domain and was not modified during the transmission of the message.

What are DKIM Selectors? The DKIM selector is specified in the DKIM-Signature header and indicates where the public key portion of the DKIM keypair exists in DNS. The receiving server uses the DKIM selector to locate and retrieve the public key to verify that the email message is authentic and unaltered.

How can I find my DKIM Selector? A DKIM selector is specified when the private/public key pair is created when DKIM is set up for the email domain (or email sender), and it can be any arbitrary string of text.

The DKIM selector is inserted into the DKIM-Signature email header as an s= tag when the email is sent. The easiest way to discover the selector for your domain is to send an email to yourself.

TODO - pridat priklady a vysledky spracovania

5.9.9 Attachments sign

In the following example (listing 59) we will show how we can sign attachments.

```
...
signPdf();//sign attachments from scenario parameters
...
```

Listing 59: ECMA/Javascript for signing attachments

For signing emails it is necessary to upload the certificate and set the custom values constants in the email (48).

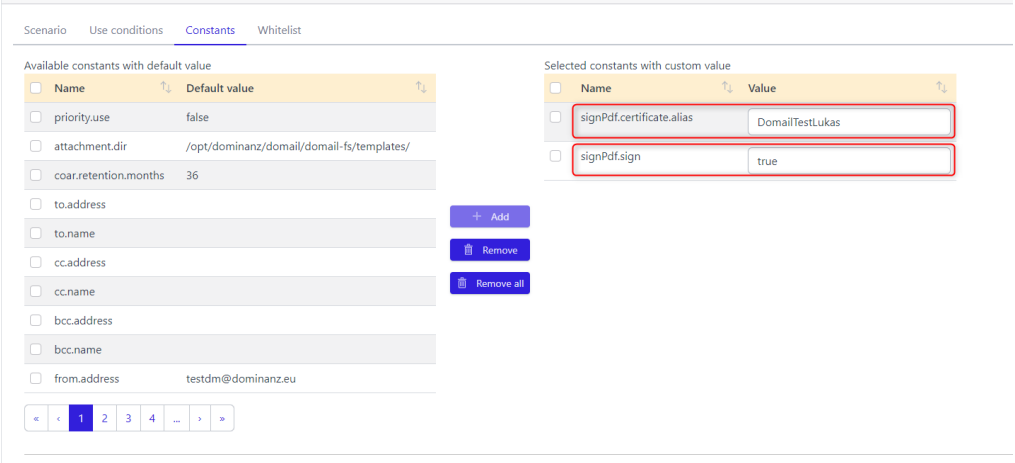

Figure 48: Scenario - set custom values constants for singing attachments

For signing is used the certificate, which was uploaded in the picture 47.
TODO: - pridat vygenerovane EML a obrazok z Outlooku po spracovani

5.9.10 Signing email

In the following example (listing 60) we will show how we can sign email.

```
...
signEmail();
...
```

Listing 60: ECMA/Javascript for signing email

For signing emails it is necessary to upload the certificate and set the custom values constants in the email (49).

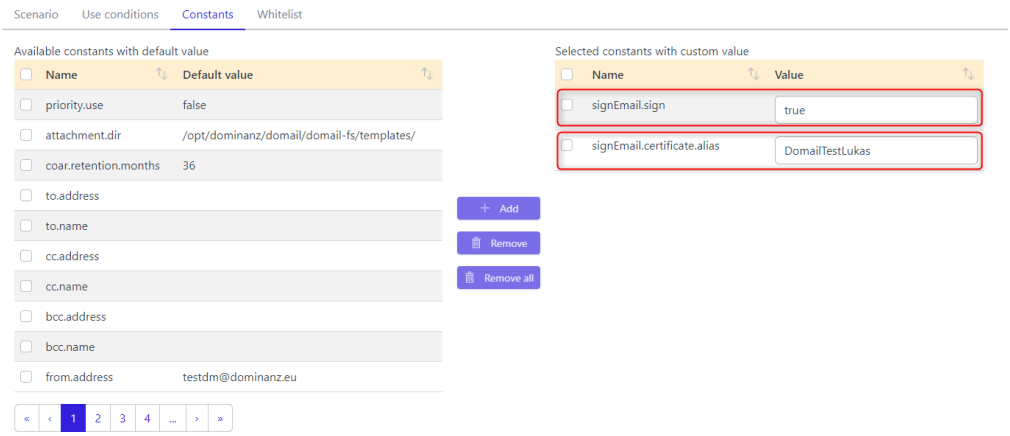

Figure 49: Scenario - set custom values constants for signing email

```
1  Date: Fri, 28 Jul 2023 06:46:39 +0000 (UTC)
2  From: doMail-test - TestVM <testdm@dominanz.eu>
3  Reply-To: doMail-test - TestVM <testdm@dominanz.eu>
4  To: NameOfEmail <lukas.vanek@dominanz.sk>
5  Message-ID: <domic.1632.1.1.597400988.5.1690526799210@domail-test>
6  MIME-Version: 1.0
7  Content-Type: text/plain; charset=us-ascii
8  Content-Transfer-Encoding: 7bit
9
10
```

Figure 50: Scenario - signing email - EML

In the following figure 51 you can see that Outlook has identified the signed delivered email.

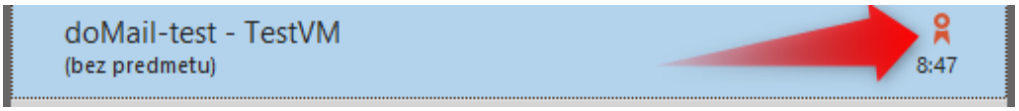

Figure 51: Scenario - signing email - Outlook

For signing is used the certificate, which was uploaded in the picture 47.

5.9.11 Context - ctx object

This class contains context objects that scroll between plugins.

- ctx.attachments;
- ctx.fail("Error message");
- ctx.getAttachmentsByExtension(".pdf");
- ctx.getDbEmail();
- ctx.getDbReq();
- ctx.getDbReqRun();
- ctx.id;
- ctx.isFailed();
- ctx.message;
- ctx.message.getMessage();
- ctx.params;
- ctx.req;
- ctx.scenarioParams;
- ctx.scriptConstants;
- ctx.service;
- ctx.service.db;
- ctx.service.fs;

5.9.11.1 attachments

Get List<Attachment<?» object - list of attachments that will be added to email after processing ends.

In the following example (listing 61) we don't send any attachment via request and we want to set the text to PlainText according to the number of attachments.

```
...
if(ctx.attachments.size() > 0){
    text.set("Mam prilohy");
}
else {
    text.set("NEMAM ziadnu prilohu");
}
...
```

Listing 61: ECMA/Javascript context - attachments

There were 0 attachments in the request, result in EML:

```
Date: Mon, 24 Jul 2023 12:58:07 +0000 (UTC)
From: doMail-test - TestVM <testdm@dominanz.eu>
Reply-To: doMail-test - TestVM <testdm@dominanz.eu>
To: Lukas Vanek <lukas.vanek+domail@dominanz.sk>
Message-ID: <domic.1599.1.1.1952104131.0.1690203487270@domail-test>
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="----=_Part_0_2018302915.1690203487282"

------=_Part_0_2018302915.1690203487282
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 7bit

NEMAM ziadnu prilohu
------=_Part_0_2018302915.1690203487282--
```

Figure 52: Context - attachments - result in EML

5.9.11.2 ctx.fail("Error message")

Mark processing as invalid with given description. In the following example (listing 62) we will show how we can failing communication.

```
...
ctx.fail("My custom error from script");
...
```

Listing 62: ECMA/Javascript for failing communication

In the following figure 53 we can see that communication is failing with our cusom error message.

General Addresses Tasks Logs Archive Original source EML source Preview

RunID	No.	Log ID	Level	Start date	Message	Origin
1	0	47724	ERROR	26.07.2023 11:31:28		MIME_PROC
1	1	47722	INFO	26.07.2023 11:31:24	Scenario for reporting is not set!	MIME_PROC
1	1	47723	ERROR	26.07.2023 11:31:26	My custom error from script	MIME_PROC

Items per page 100

Figure 53: Context - failing communication

5.9.11.3 ctx.getAttachmentsByExtension(".pdf")

Get List<Attachment<?» object - list of attachments that will be added to email after processing ends that have specified extension in their name.

5.9.11.4 ctx.getDbEmail()

Get srv_emails object - database entry of single email

5.9.11.5 ctx.getDbReq()

Get srv_emlreq object - database entry of request

5.9.11.6 ctx.getDbReqRun()

Get srv_procrun object - database entry of run

5.9.11.7 ctx.id

Get IdNumRunid object - id of actual processing composed of id of request, num - index of email and runid - id of run

5.9.11.8 ctx.isFailed()

Returns true if error ocurred during processing, otherwise return false

5.9.11.9 ctx.message

Get MailMessage object for work with generated message

5.9.11.10 ctx.message.getMessage()

Get SMTPMessage object - generated email

5.9.11.11 ctx.params

Get Map<String, Object> object - map of parameters set during processing

5.9.11.12 ctx.req

Get CreateCommunicationRequestI object - request to send email.

In the following example (listing 64) we will show how we can replace a text in plain text body of email. Result for this request JSON (listing 63):

```
{
    "to": [
        {
            "addressName": "Lukas Vanek",
            "emailAddress": "lukas.vanek+domail@dominanz.sk"
        }
    ],
    "plainTextBody": "Povodny text emailu v systeme $domail$",
    "scenario": "Lukas-SOAP/REST",
    "testmode": false
}
```

Listing 63: Example for POST request for replacing text in body - PLAIN

In the following example (listing 64) we will show how we can replace a text in plaing text.

```
...
var plainText = ctx.req.getPlainText();
if(plainText != null && plainText != ""){
    plainText = plainText.replace('$domail$','Domail');
    text.set(plainText);
}
...
```

Listing 64: ECMA/Javascript for replacing text in body - PLAIN

Result in EML:

```
Date: Wed, 26 Jul 2023 10:13:18 +0000 (UTC)
From: doMail-test - TestVM <testdm@dominanz.eu>
Reply-To: doMail-test - TestVM <testdm@dominanz.eu>
To: Lukas Vanek <lukas.vanek+domail@dominanz.sk>
Message-ID: <domic.1621.1.1.244258411.0.1690366398293@domail-test>
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="----=_Part_0_1901670238.1690366398307"

------=_Part_0_1901670238.1690366398307
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 7bit

Povodny text emailu v systeme doMail
------=_Part_0_1901670238.1690366398307--
```

Figure 54: Context - req - replace text in EML

5.9.11.13 ctx.scenarioParams

Get Map<String, String> object - map of parameters set for actual scenario

5.9.11.14 ctx.scriptConstants

Get Map<String, String> object - map of set constants

5.9.11.15 ctx.service

Get services to work with

5.9.11.16 ctx.service.db

Get DbService object - service for work with database

5.9.11.17 ctx.service.fs

Get FsService object - service for work with file system

Alphabetical Index